M.S. THESIS

# Predicting live migration performance of virtual machines using machine learning

기계 학습을 통한 가상화 플랫폼의
라이브 마이그레이션 성능 예측

FEBRUARY 2016

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Jinho Song

# Predicting live migration performance of virtual machines using machine learning

지도교수 Dr. Bernhard Egger

이 논문을 공학석사학위논문으로 제출함

2015 년 10 월

서울대학교 대학원

컴퓨터 공학부

송 진 호

송 진 호의 석사학위논문을 인준함

2015 년 12 월

위 원 장 : 문 병 로 (인)

부위원장 : Bernhard Egger (인)

위 원 : Srinivasa Rao Satti (인)

# Abstract

## Predicting live migration performance of virtual machines using machine learning

Jinho Song

Department of Computer Science and Engineering

College of Engineering

The Graduate School

Seoul National University

Virtualization is a widely used technology these days as most of server computing environments are rapidly shifting to cloud computing. Live migration, one of the most compelling features in system virtualization, has been an active area of research. Attempts to predict migration performance were made, but most of those were limited to analytical approaches with relatively unstable prediction errors or not easy to extend to realistic environments as more parameters are identified and considered. In this thesis, a novel data driven approach based on the support vector regression method providing flexibility and extensibility in parameter selection is introduced to predict performance metrics such as total migration time, downtime and the total amount of transferred data, especially on QEMU which is hardware

virtualization platform that is open-source and the method of this thesis is easy to adapt to various purposes. It will facilitate automated system administration with live migration more efficiently.

**Keywords : virtualization, live migration, machine learning, support vector machine**

**Student number : 2007 - 21003**

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

System virtualization is a widely used technology in cloud computing environments providing system administrators resource management, server consolidation, load balancing and system availability. It serves as the abstraction for the physical resources and applications so that available resources such as storage, application, server and network devices can be shared between hosts according to the usage consumption rate in virtualized computing environment. Thanks to elasticity and scalability in cloud computing service, OS host applications can increase or decrease their resource usage amount on operational needs and user hosted services can be switched between physical hosts without service interruption perceived.

One of the most powerful and popular features of system virtualization in cloud computing environment is live migration, i.e., moving the entire execution environment from one physical host to another without or with minimum service interruption. This is very important to system administrator because it makes the level of service agreement (SLA) committed by service providers fulfilled as high

as 99.999% or higher, which means less than 5 minutes of downtime in a year. [1] VM migration makes servers with workload overloaded or overheated balanced dynamically to overcome physical host capacity limitation and manages servers that needs to be selectively brought down for maintenance after migrating their workloads to other servers [2]. For these reasons, latest virtualization solutions such as Xen and VMware have already built and embedded these live migration functions into their implementation, called XenMotion[3] and Vmotion respectively.

Although live migration is such an attractive feature in virtualization environment and many researches have been made to induce prediction models of live migration performance, those previous works are either limited to a small set of well-known parameters or evaluated on specific solutions only and still seem to be difficult to extend to generalized cases. Many modeling approaches have been presented to provide prediction for live migration performance for which total migration time and downtime are key metrics, but those are mostly analytical model approaches and still have limitations in prediction accuracy and its variance. [4][5]–[11]

In this thesis, parameters affecting each live migration algorithm most are assumingly listed and evaluated using data-driven modeling approach. The support vector regression method in machine learning is used along with more enhanced features such as bagging for better performance prediction as well. The results show that prediction errors are lower than previous analytical or empirical methods, especially with low variance. In addition, it provides extensibility and flexibility to system administrators for virtualized computing environment.

This thesis is organized as follows : In Section 2 and 3, live migration algorithms which are being commonly used are presented and what metrics are affecting performance evaluation along with existing models and evaluation attempts. In Section 4, data driven approach is introduced to be evaluated in real application scenarios and get more accurate prediction rates. Experimental results are explained in Section 5. Finally, in Section 6, the conclusion is drawn with still challenging areas explaining where more research is necessary to improve live migration performance.

# Chapter 2

# Background and related work

Live migration is a technology transferring system states of an entire running VM from one physical host to another. System states including active memory and execution state are transferred from the source to the destination machine without perceivable interruption in service availability. For example, when migration is complete, physical system resources such as virtual I/O devices are disconnected from the source and re-directed to the destination physical host under a very short down time to make the service running on the host available to users seamlessly. There are two main approaches in live migration methods : pre-copy migration and post-copy migration. They differ depending on when the state is transferred, i.e. before or after VM execution is switched. Pre-copy method, which is more widely implemented and used in most VM hypervisors, has many variations in ways to deal with resource management or with regards to performance characteristics of live migration.

## 2.1 Live migration algorithms

Live migration algorithm consists of steps such as 1) transfer dirty pages, 2) suspend at source, 3) transfer last pages, 4) resume at destination. Depending on when the step #2 is performed, it is broken into two distinctive approaches as pre-copy and post-copy memory migration. Post-copy method first suspends the migrating VM at the source before transferring dirtied memory pages whereas pre-copy stops the VM after copying memory pages with VM running on the source host.

### 2.1.1 Pre-copy migration

The main idea of pre-copy migration is transferring system state iteratively and minimizing subsequent stop-and-copy phase between migration hosts. Pre-copy live migration is performed in the following steps :

All memory pages are marked as dirty indicating changes in system states. On every iteration, the memory pages that are dirtied in the source host during the previous iteration are resent to the destination host so that the system states get synchronized on both sides. When the number of memory pages transferring in the source host goes below a specified criteria, i.e. when the number of dirtied pages are small enough to stop the VM, which is primarily influenced by the network bandwidth and the preferred downtime, the VM is suspended at the source host and the remaining pages are transferred to the target host to complete the migration. As a result, the VM is resumed on the destination.

Pre-copy migration process includes 6 distinctive stages as follows:[3]

1) Pre-stage : a target is pre-selected so that the resources required to receive migration can be guaranteed.
2) Reservation: resources at the destination host are reserved.
3) Iterative pre-copy: pages dirtied during the previous iteration are sent to the destination. The entire memory is sent in the first iteration.
4) Stop-and-copy: the VM is stopped temporarily for a final transfer iteration.
5) Commitment: the destination host confirms that it has received a consistent copy of the VM.
6) Activation: resources are re-attached to the VM on the destination host.

Stop conditions determine when it is the right time for the stage to terminate. If there are no stop conditions, the iterative stage may continue endlessly. These conditions which are affected by the design of both the hypervisor and the live migration subsystem, are important in reducing the amount of data copied between physical hosts while minimizing VM downtime. The existence of these stop conditions, however, has a significant effect on migration performance and thus may cause non-linear trends in the total migration time and downtime.

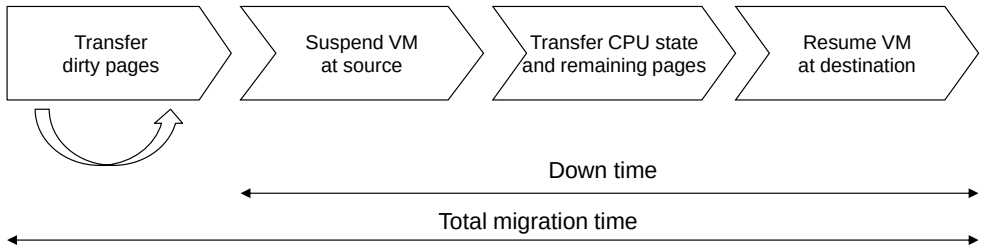



Figure 2.1 Pre-copy migration

### 2.1.2 Post copy migration

Hines et. al [12] presented the design, implementation, and evaluation of post-copy based live migration for virtual machines (VMs) across a Gigabit LAN. Post-copy, in contrast to pre-copy live migration approach, is to move the VM execution state to the destination host at the beginning of the migration process and the memory pages are sent as requested by the VM. Figure 2.2 shows how the process is performed. Post-copy approaches are intended to solve the predictability of total migration time and reduce downtime with pre-copy migration, but as the pages have to be requested over the network even before the VM has access to them, VM and its applications experience performance degradation when the VM is resumed although a novel attempt was tried to reduce performance penalty for retrieving pages in post-copy using Remote Direct Memory Access, RDMA.

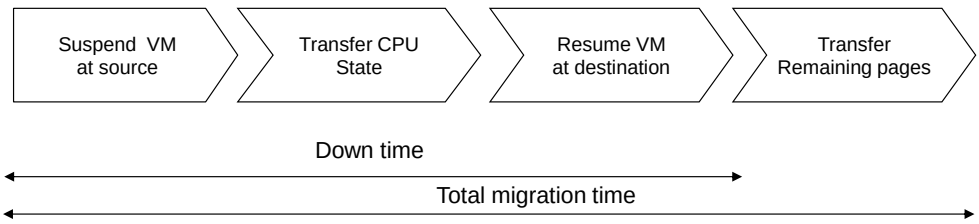



Figure 2.2 Post-copy migration

### 2.1.3 Compression algorithm

Original pre-copy live migration was difficult to perform rapid migration with low network overhead due to a great amount of transferred data during migration. Jin et. al.[13] designed and implemented a novel memory-compression based VM migration algorithm called MECOM. Memory compression approach was the first

attempt to facilitate fast and reliable virtual machine migration while virtual machine services are not so much affected by memory page characteristics. Although compression helps network bandwidth to increase its availability and takes much less time in transferring compressed dirty pages, it is not sufficient to apply to server instances with huge VM size yet.

### 2.1.4 XBZRLE algorithm

When migrating VM with high workloads or low network bandwidth, it is very probable to encounter service interruption when VM memory pages are dirtied faster than they are transferred over the network, which means that it leads to extended migration downtime. In order to solve this issue, delta compression approach was presented by Svard et. al.[14]. They designed and implemented delta compression live migration algorithm as a modification to the KVM hypervisor and evaluated its performance by migrating running VMs with different type of workloads. The result showed a significant decrease in migration downtime. XOR binary RLE(Run Length Encoding) live migration algorithm was adopted as a compression algorithm. Delta compression is performed as follows:

When transferring a page, if the cache has a previous version of the page in the source host, a delta page from the changes between the new version and the cached version is made using XOR operations. The delta page is compressed through RLE then and the compressed page is transferred after the cache is updated. Likewise on the destination side, the delta page is decompressed and the page is recreated from the delta page using XOR.

Delta compression is a definitely key algorithm when migrating large VMs in practical perspectives and identifying parameters that affect its performance is very important in provisioning and relocating of VMs in cloud infrastructure.

### 2.1.5 Auto-converge algorithm

As a way of live-migrating virtual execution environment in wide-area network, pre-copying with write throttling approach was presented by Bradford et al. [15]. The algorithm is also called a dynamic rate limiting technique, i.e., the amount of hardware resources allocated to the migration task increase dynamically at the expense of the performance of the VMs. An entire running web server, including its local persistent state, with minimal service interruption can be transferred within 3 seconds in the LAN and 68 seconds in the WAN environment.

## 2.2 Performance metrics

Many parameters are known to be affecting migration performance as shown in Table 2.1. Page dirty rate is the most influencing one in any pre-copy variants among those, because otherwise frequent dirtying of memory page will get migration job set back continuously. VM size and writable working set size are also commonly referred parameters that are influencing in pre-copy algorithm. VM size is the total amount of memory allocated to VM and sets lower bound of number of pages to transfer because the total amount of transferring data increases as more pages are dirtying. By the way, VM size is notably the only factor that affects performance of post-copy algorithm which has nothing to do with memory changes. Write density rate imposes direct impact on delta compression algorithm, so it should be taken into account when assessing performance evaluation as well. The rest of parameters that affect migration performance are listed in Table 2.1.

Managing performance overhead of live migration is very important in system administration of virtualized computing environment and many researches have been made to define and model performance metrics. [2][16][17]. The following list of metrics have been commonly used to measure and predict the performance of live migration [12] :

1. Downtime: The time between pausing the VM on the source and resuming it on the destination
2. Total Transferred Data: The total amount of memory pages transferred, including duplicates, throughout all of the whole migration .
3. Total Migration Time: Total sum of times spent during of all the migration stages. Total time is very important because it is tightly coupled with resource usage on both of nodes.
4. Performance Degradation: The extent to which migration affects application performance within the VM such as service availability or responsiveness to end user

In this research, downtime, total migration time and total transferred data are chosen and evaluated for performance prediction.

Table 2.1 Parameters and performance metrics

| **Parameters** | **Performance metrics** |
|---|---|
| · Page Dirty Rate<br>· VM Size | · Total Migration Time<br>· Downtime |

| Parameters | Performance metrics |
| --- | --- |
| · Writable Working Set Size<br>· Write Density<br>· Non Writable Working Set Size<br>· Working Set Entropy<br>· Non Working Set Entropy<br>· Unhalted Cycles<br>· Retired Instructions<br>· Cache Misses<br>· Cache References<br>· Cache Hit<br>· IPC<br>· L2$ WB Count<br>· Storage NIC Utilization<br>· Available CPU resource on Host | · Total Transferred Data |

## 2.3 Existing models and evaluation attempts

Although researches have been made to define and evaluate models to predict live migration performance, most of which were analytical models, they were designed

with regards to an individual parameter, not in a conjunctive fashion, hence not easy to extend their application boundaries to general cases as more parameters are to be considered. Moreover, their approaches were limited to original pre-copy migration on Xen only and yet on the way to model other algorithms. Data driven approaches were also made to build performance prediction models, but those are resorting only to legacy modeling methods such as power regression or model checker method, so not applicable to general migration cases in reality. [18] [19]

# Chapter 3

# Empirical evaluation

Empirical evaluations were performed with regard to five different types of live migration algorithms supported on QEMU platform for the following reasons : to assume parameter set that affects live migration performance most, to generate training sample data to evaluate those assumed parameters, finally to build a performance prediction model with them.

## 3.1 Sample generation and evaluation

Parameter samples were generated and collected using monitoring functions provided by KVM/QEMU virtual machine hypervisor. Four pre-copy types and 1 post-copy type algorithms are supported on KVM/Qemu and migrations with those algorithms were performed to evaluate performance parameters. Each parameter was assessed by turning on VM options or applying methods as follows:

Page dirty rate data was generated by enabling global_dirty_log option in QEMU: Dirtied bitmaps are cleared every 100ms to calculate page dirty rate. (Figure 3.1) : Performance monitoring tool for Linux call ‘perf’ is also used with command line option

- `perf stat -e mem-stores -p [PID] -I [print interval in ms]`

Each host machine has 3 NICs for VM service, VM management and shared storage respectively. VM runs on the machine with 4 CPU and 2 GB memory configuration.

## 3.2 Workloads

Real application benchmarks were applied to make various types of workloads and identify characteristics of each live migration techniques. The list of benchmark workloads is listed in Table 3.1

Table 3.1 Benchmark Workloads

| **Benchmark** | **Tool Configuration** |
|---|---|
| Parsec | canneal, facesim, fluideanimate, freqmine, raytrace, streamcluster, swaptions |
| Dacapo | avrora, eclipse, fop, h2, jython, luindex, pmd, sunflow, tomcat, tradebeans, tradesoap, xalan |
| OLTP | auctionmark, epinions, tatp, tpcc, twitter, ycsb |

| Benchmark | Tool Configuration |
|---|---|
| benchmarks | |
| Mplayer | Valkaama 720p, Valkaama 1080p, Tears of Steel 1080p |
| Bzip2 | Compression of Wikipedia dump data |

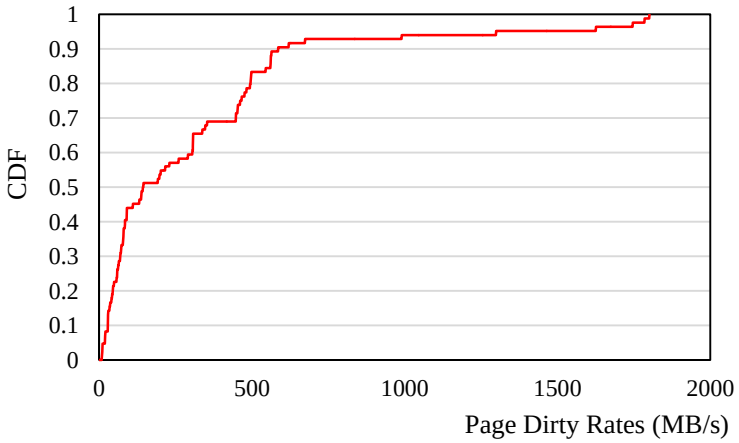
For parsec benchmark generation, total 28 workloads are applied with 7 applications and 4 threads at the maximum. Various VM sizes with random reboots during experiment were tested.

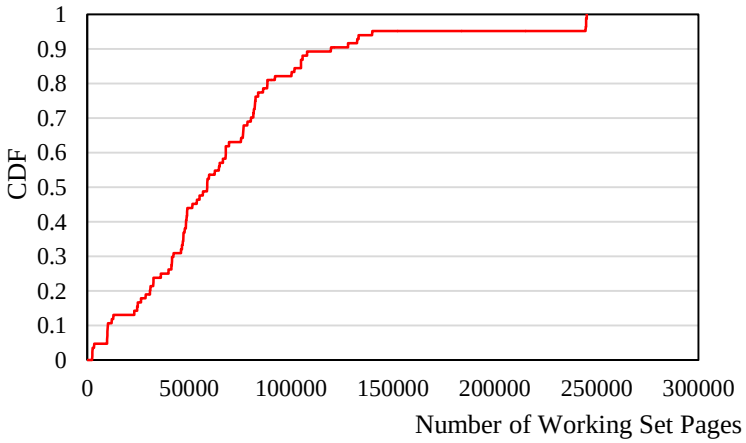
For Dacapo benchmark, total 48 workloads were made using 12 applications with 4 threads at the maximum. In OLTP workload, 5 trials were made with variations in start time and found that cache hit ratio of DB was affected by warm-up time. Total 12 workloads were made for OLTP. Performance degradation could be measured during this generation.

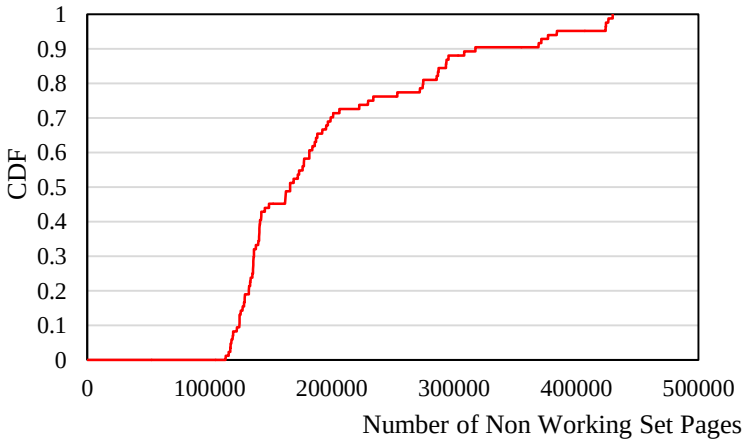
Standalone client applications such as ‘mplayer’, ‘bzip2’ and ‘make’ were used to find application specific characteristics. Open source movies were used for respective resolution option. (Valkaama 720p, Valkaama 1080p, Tears of Steel 1080p). Mplayer application was revised to print a message whenever a frame is decoded so that performance degradation could be checked accordingly. For bzip2, dump data from UK Wikipedia was used. Data was generated for 9 different compression levels. The ‘make’ workload was generated by compiling Linux kernel 4.2.3 with 5 threads at the maximum. A total of 104 workloads were applied.

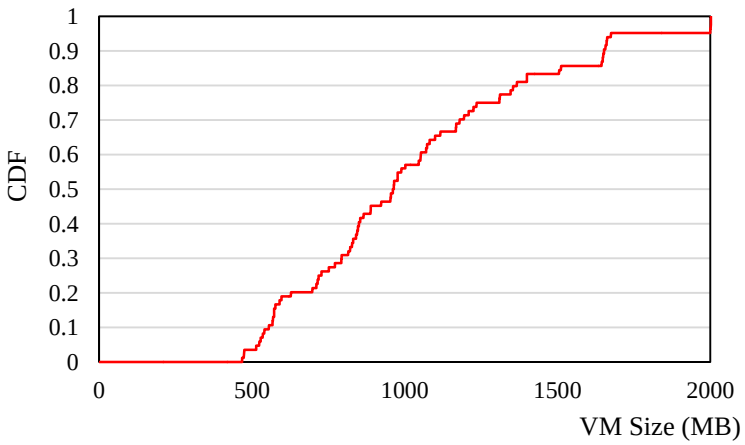
Figure 3.1 shows workload statistics of performance parameters as CDF(cumulative density function) graphs used in this research. For example, the

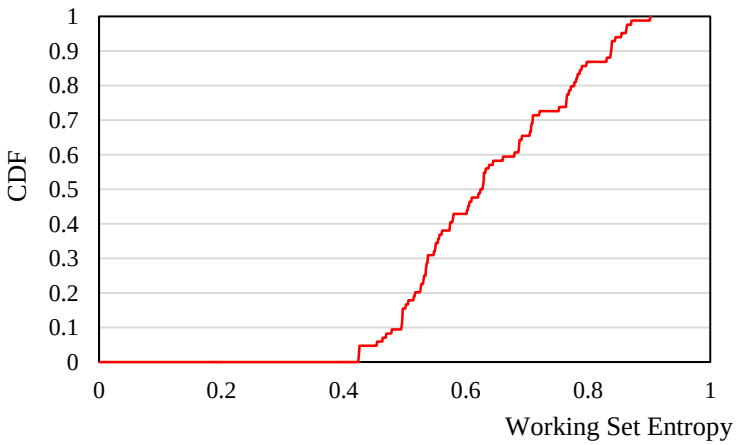
CDF regarding VM size shows the allocated physical memory to a VM with 2 GB of virtual memory. Although the average write density rate and non-working set entropy have dense distributions around specific ranges, the data sets for most of the candidate parameters show quite evenly distributed and seem sufficient to be chosen for modeling.

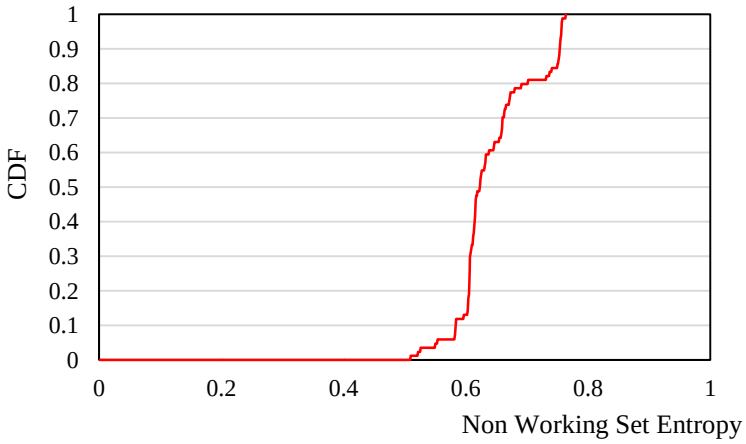


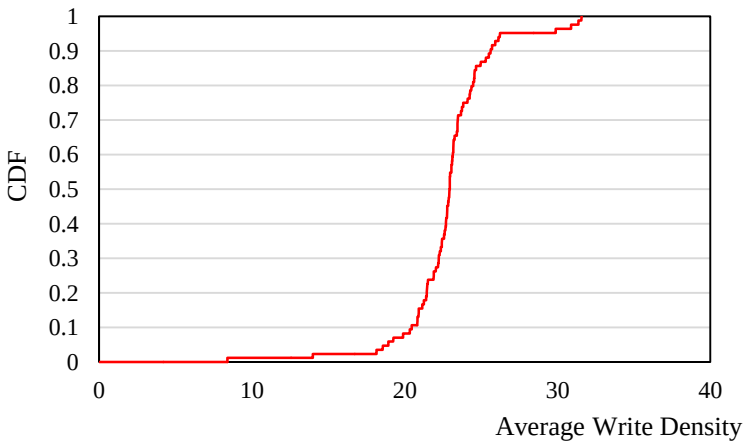




CDF
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
100000
200000
300000
400000
500000
Number of Non Working Set Pages


CDF
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
VM Size (MB)


CDF
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Working Set Entropy



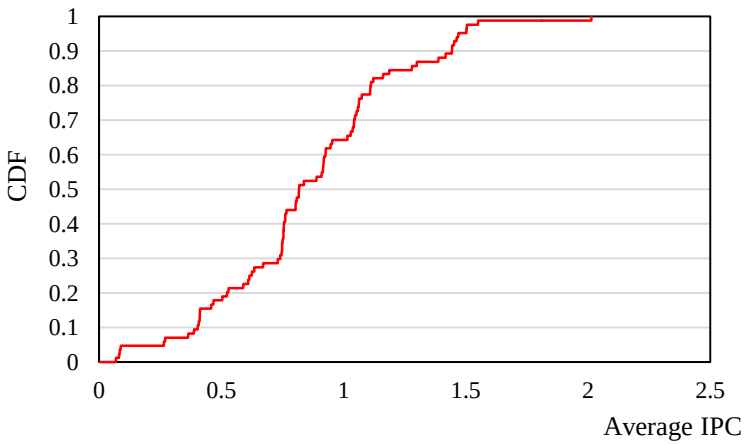
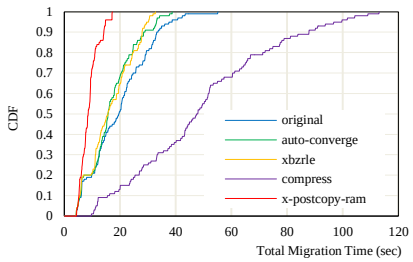




Figure 3.1 Workload statistics for each parameter

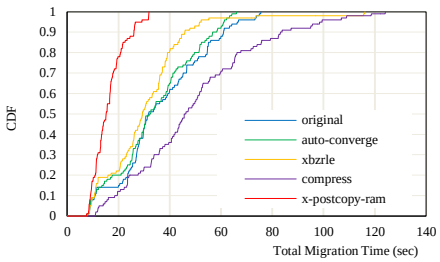
Live migration results with those workloads applied are shown in Figure 3.2. As in the graphs, the total migration time shows a more dense distribution with higher network bandwidth values assumingly because it has many other affecting variables on lower network bandwidths. The post-copy algorithm is directly influenced by the network bandwidth which relates with the amount of data while the compress algorithms, however, are not being affected by the network bandwidth because the CPU overhead prevents network resources from being utilized fully.

In contrast to the total migration time, the downtime shows that it has a properly uniform distribution and does not get affected by the network bandwidth with the exception of the post-copy algorithm which has a narrow range of distribution.

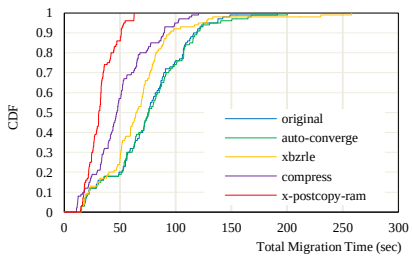
The total amount of data transferred makes a quite clear distinction between post-copy/compression and other algorithms.



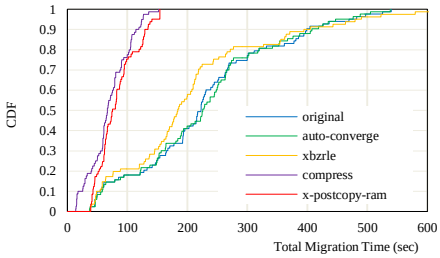

1,000 Mbit/s




750 Mbit/s

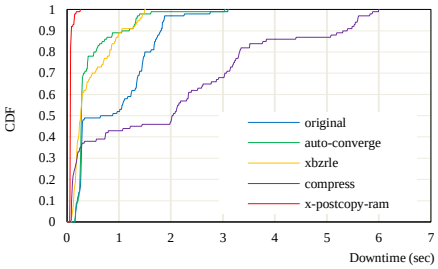



500 Mbit/s

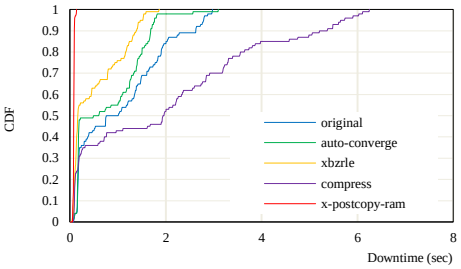



100 Mbit/s

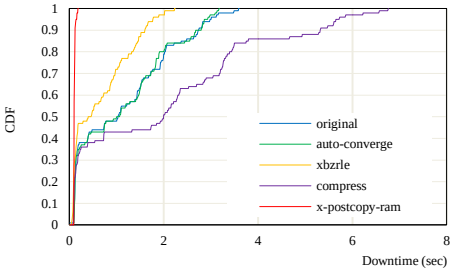
a) Total Migration Time with network bandwidth 1,000 / 750 / 500 / 100 Mbit/s respectively



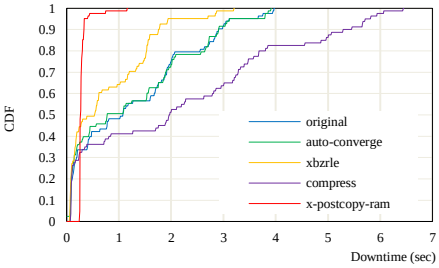

1,000 Mbit/s




750 Mbit/s

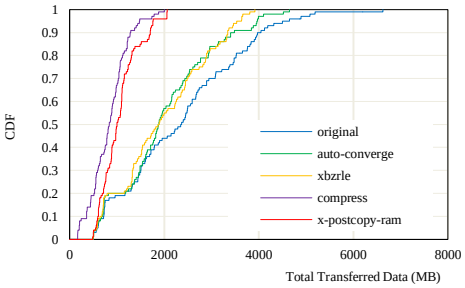



500 Mbit/s

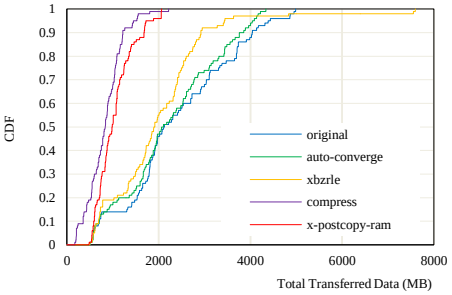



100 Mbit/s

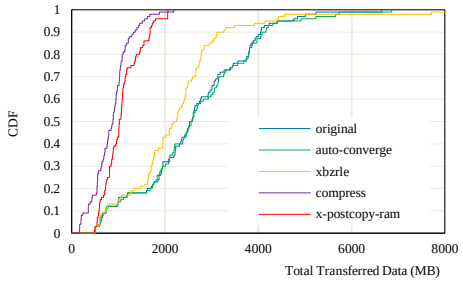
b) Downtime with network bandwidth 1,000 / 750 / 500 / 100 Mbit/s respectively



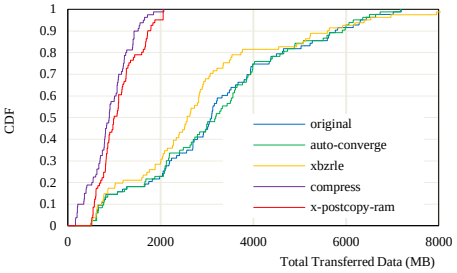

1,000 Mbit/s




750 Mbit/s

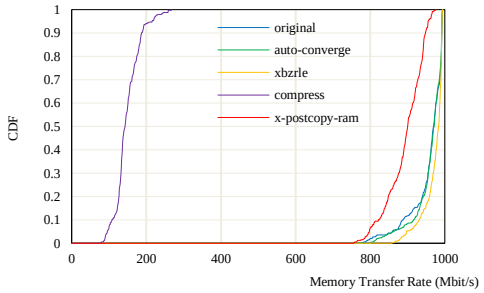



500Mbit/s

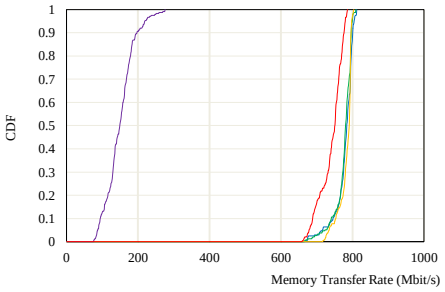



100 Mbit/s

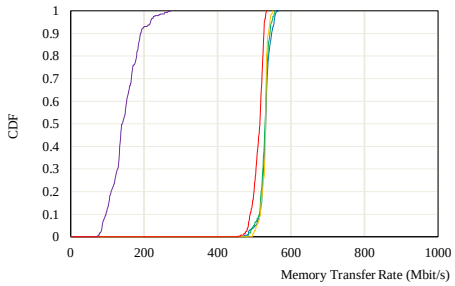
c) Total Tranferred data with network bandwidth 1,000 / 750 / 500 / 100 Mbit/s respectively



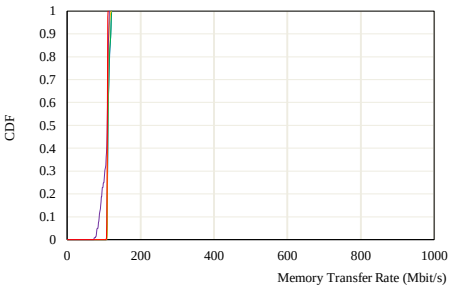

1,000 Mbit/s




750 Mbit/s




500Mbit/s




100 Mbit/s

d) Network Throughput with network bandwidth 1,000 / 750 / 500 / 100 Mbit/s respectively

Figure 3.2 Live migration results for a) Total time, b) Downtime, c) Total transferred data, d) Network Throughput

As shown in Table 3.2 that lists the coefficient of determination for the parameters, the assumed parameters affect migration performance. In particular, the average dirty rate is the most dominant factor for performance throughout for the five different migration algorithms in concern while the entropy is observed to be less affecting, although it has minor effects on the compression algorithm .

Table 3.2 R^2 of parameters for performance metrics

| | | DR | SIZE | WSS | NWSS | WSE | NWSE | L2$_WB | WRD | STRGU | CPUSYS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Total Time | vanilla pre-copy | **0.17** | **0.48** | **0.54** | 0.23 | 0.14 | 0.17 | 0.16 | 0.09 | 0.23 | **0.31** |
| | cpu-throttling | **0.20** | **0.50** | **0.52** | 0.22 | 0.13 | 0.17 | **0.18** | 0.12 | 0.27 | **0.27** |
| | delta-compression | **0.08** | **0.34** | **0.31** | 0.24 | 0.06 | **0.21** | **0.13** | 0.11 | 0.23 | **0.10** |
| | data-compression | **0.24** | **0.69** | **0.57** | **0.44** | 0.19 | 0.19 | **0.24** | 0.15 | 0.05 | **0.16** |
| | post-copy | 0.09 | **0.99** | 0.26 | 0.86 | 0.28 | 0.17 | -0.02 | 0.19 | 0.05 | 0.09 |
| Downtime | vanilla pre-copy | 0.72 | 0.41 | 0.67 | 0.21 | 0.30 | 0.16 | 0.48 | 0.25 | -0.01 | 0.29 |
| | cpu-throttling | 0.71 | 0.38 | 0.67 | 0.27 | 0.28 | 0.10 | 0.47 | 0.25 | 0.01 | 0.22 |
| | delta-compression | 0.62 | 0.36 | 0.58 | 0.19 | 0.07 | -0.16 | 0.43 | 0.28 | -0.18 | 0.09 |
| | data-compression | 0.72 | 0.32 | 0.58 | 0.27 | 0.21 | 0.12 | 0.52 | 0.19 | 0.06 | 0.21 |
| | post-copy | -0.04 | -0.02 | -0.04 | -0.03 | -0.01 | -0.03 | -0.04 | -0.04 | 0.04 | -0.04 |
| Transferred Data | vanilla pre-copy | 0.21 | 0.47 | 0.58 | 0.20 | 0.14 | 0.16 | 0.19 | 0.10 | 0.19 | 0.34 |
| | cpu-throttling | 0.23 | 0.49 | 0.57 | 0.20 | 0.13 | 0.17 | 0.21 | 0.13 | 0.23 | 0.29 |
| | delta-compression | 0.10 | 0.34 | 0.33 | 0.24 | 0.06 | 0.20 | 0.15 | 0.12 | 0.21 | 0.11 |
| | data-compression | 0.23 | 0.65 | 0.57 | 0.43 | 0.19 | 0.17 | 0.21 | 0.13 | 0.02 | 0.19 |
| | post-copy | 0.09 | 0.99 | 0.26 | 0.86 | 0.28 | 0.17 | -0.02 | 0.19 | 0.05 | 0.09 |

# Chapter 4

# Data driven approach

In this section, a novel, machine learning based data-driven approach is introduced to analyze and estimate live migration performance. Sets of training samples are generated for selected parameter sets which are assumed to affect live migration performance and used for regression analysis to build performance models using support vector machine.

## 4.1 Parameter selection and migration algorithms

Profiling data is collected by executing VM migration with 4 pre-copy and 1 post-copy algorithms and used again to build models based on machine learning approach.

Table 4.1 shows the list of parameters chosen to build model for each live migration algorithm respectively.

Table 4.1 Parameter selection for each algorithm

| Algorithm | Writable working set size | Page Dirty Rate | VM size | Entropy | Avg. Wr. Density | L2 cache writeback count | Non-halted cycles |
|---|---|---|---|---|---|---|---|
| Pre-copy | O | O | O | X | X | X | X |
| XBZRLE | O | O | O | X | O | X | X |
| Auto converge | O | O | O | X | X | O | O |
| Compress | O | O | O | O | X | X | X |

## 4.2 Prediction using support vector regression

Support vector machine is a supervised learning method for efficient model classification and regression that was first introduced by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. [20] Once training examples are given, each sample is marked to fall into one of two categories and an SVM training algorithm builds a model that maps new samples into one category or the other, making it a non-probabilistic binary linear classifier.

Examples are represented as points in space in a SVM model, mapped so that the examples of the separate categories are divided by a clear distance gap that is as wide as possible called a decision line. Any new examples are then mapped into that

same space and predicted to be in a category depending on which side of the gap they are close to. Not only in linear classification, SVMs can also efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces[21].

The idea of non-linear SVMs is that the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable. If every data point mapped into high-dimensional space via some transformation $\Phi = \chi \rightarrow \phi(\chi)$, the inner product becomes:

$$\mathrm{K}(x_i, x_j) = \phi(x_i)^{\tau}\phi(x_j)$$

With kernel functions, non-separable problem can be made separable by mapping data into better representational space. Among commonly used kernel functions, RBF kernel(Radial basis function) is adopted for kernel trick.

SVR function is $f(\mathbf{x}) = \sum_i (\alpha_i - \alpha_i^*) k(\mathbf{x}_i, \mathbf{x}) + b$. To find the unknown parameters of the SVR function, the following function needs to be solved.

$$\min_{\mathbf{w},b,\xi_i,\xi_i^*} \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_i (\xi_i + \xi_i^*) \quad \text{subject to} \quad \begin{array}{ll} y_i - (\mathbf{w}'\phi(\mathbf{x}_i)) - b \leq \epsilon + \xi_i \\ (\mathbf{w}'\phi(\mathbf{x}_i))) + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \qquad i = 1, 2, \ldots, n \end{array}$$

RBF kernel is $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{\Phi}(\mathbf{x}_j) = e^{-\gamma \| \mathbf{x}_i - \mathbf{x}_j \|^2}$

where optimal C, $\epsilon$, and $\gamma$ values can be found heuristically from training samples.

## 4.3 Tool architecture

The training and prediction process to build a model using support vector regression method is shown in Figure 4.1. The former half of the process is for sample generation and the latter half covers building prediction models with those generated sample data.

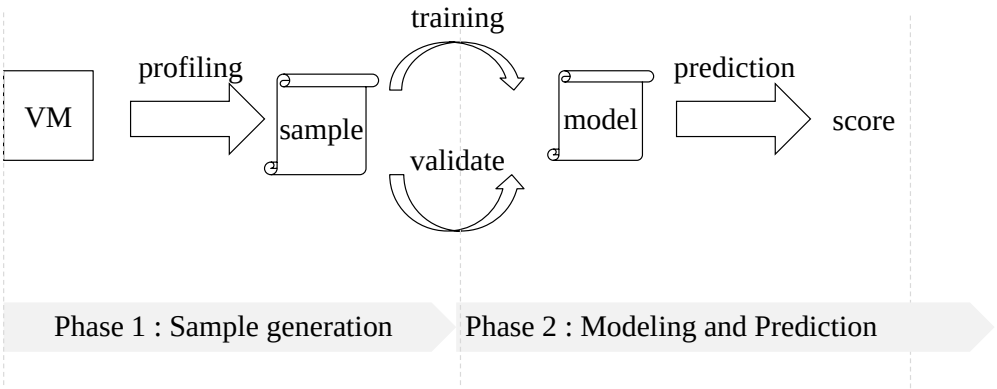



Figure 4.1 Process for prediction model using profiling samples

Profiling data is generated and put into sample database which is in turn being used for training and validation samples to build a model. In this research, the existing tool called LMbench providing integrated analysis and prediction was revised to support machine learning method. The overall architecture is depicted as in Figure 4.2.

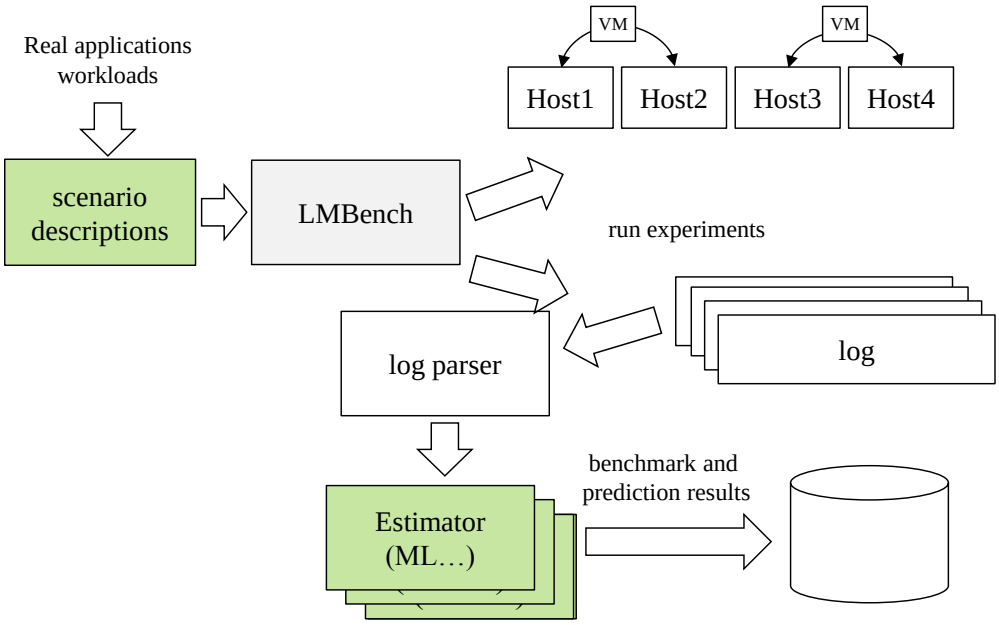



Figure 4.2 LMbench architecture

## 4.4 Single vs. multiple predictors

Not only the basic learning model where only a single predictor is generated, ensemble methods allowing multiple predictors were also used for improvement in accuracy and stability. In this conjunctive mode, individual models from original training dataset are combined to generate the final model as shown in Figure 4.3. Even with small set of training samples, combinative modeling can make remarkable results by bootstrapping those samples. One of the ensemble methods, bootstrap aggregating, also called bagging [22] is employed to show improvement in stability and accuracy to support vector regression approach in this research.

For bagging, let L be a training set $\{(x_i, y_i) \mid x_i \text{ in } X, y_i \text{ in } Y\}$, drawn from the set $\Lambda$ of possible training sets from parameter evaluations. A predictor $\Phi : X \rightarrow Y$ is a function that for any given x and it produces $y = \Phi(x)$. A learning algorithm is $\Psi : \Lambda \rightarrow \Phi$ where given any L in $\Lambda$, it produces a predictor $\phi = \Psi(L)$ in $\Phi$. This predictor generation is repeated $N$ times and for each bootstrap sample $L_k$ from $L$, a predictor is trained using $L_k$.

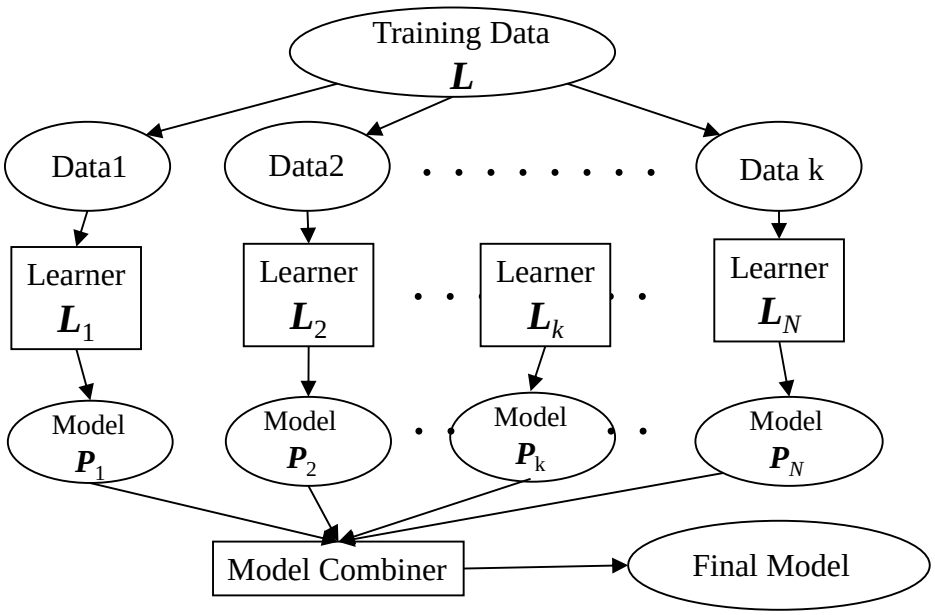



Figure 4.3 Ensemble methods combining multiple predictors

# Chapter 5

# Experimental evaluation

## 5.1 Training setup

A total of 10,000+ samples were generated from real applications benchmark and used for training. For a singular predictor, leave-one-out cross validation that was suitable for small dataset was used to validate performance model, where all samples except for validation samples are used to train model and validate the model on the target samples. Feature scaling is also applied as pre-processing of data because samples data is assumed to be scaled in SVR.

The number of samples for each predictor started from 10 with pre-defined step 5 to find out threshold values in bootstrapping with multiple predictors. Optional values used for kernel function were chosen with static values because those are not affecting results much across the whole set of samples. Thus, C for 1.0, epsilon for 0.1 and gamma for 0.0 were used uniformly respectively.

## 5.2 Prediction results

Figure 5.1 – 5.5 show prediction results from SVR models. Each graph has four lines with regard to each algorithm where score and accuracy values for single predictor and multiple ones (bagging) respectively. Throughout the whole charts, score and accuracy values are raised from around 10 samples and approximate to upper boundary around 50 or more samples even though more than 50 samples are tested for the specific algorithm. While prediction error rates were expected to go down with bagging(bootstrap aggregation) methods, it remained the same or slightly higher values because bagging is known to show slight degradation in performance when used for stable algorithm or data set.

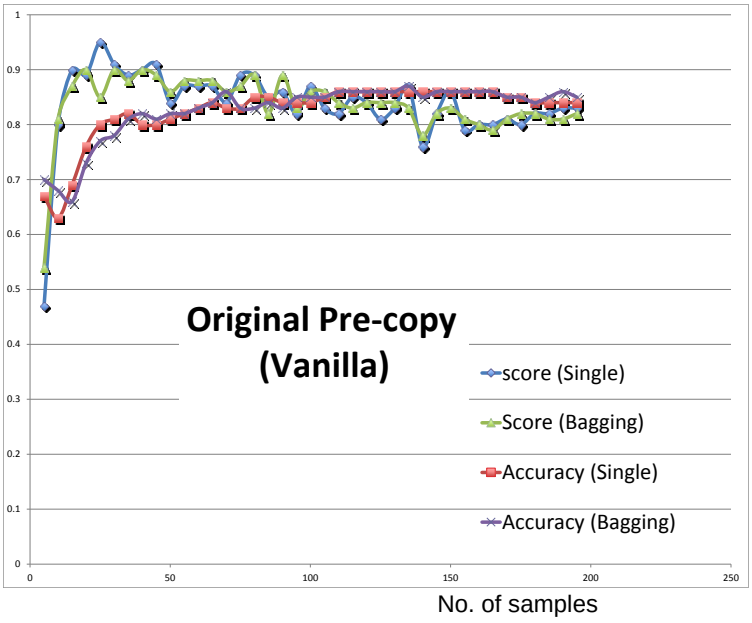



Figure 5.1 Training score and prediction accuracy for original pre-copy

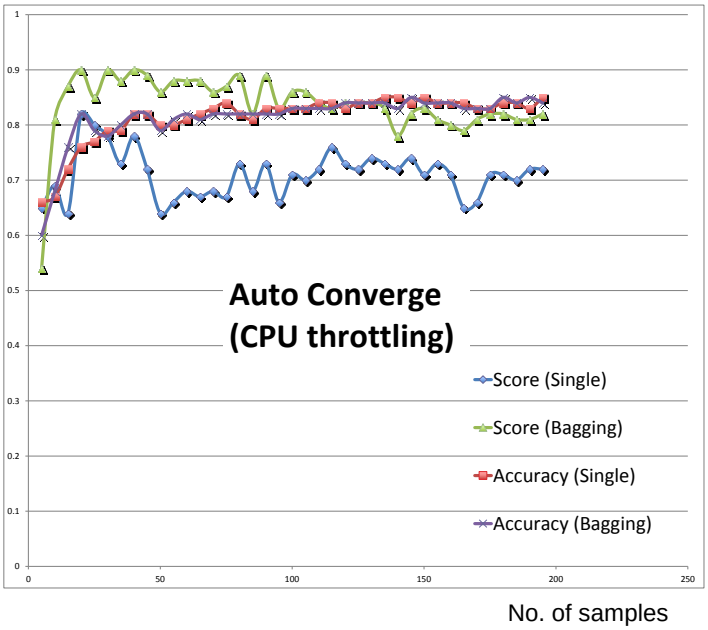



Figure 5.2 Training score and prediction accuracy for auto-converge

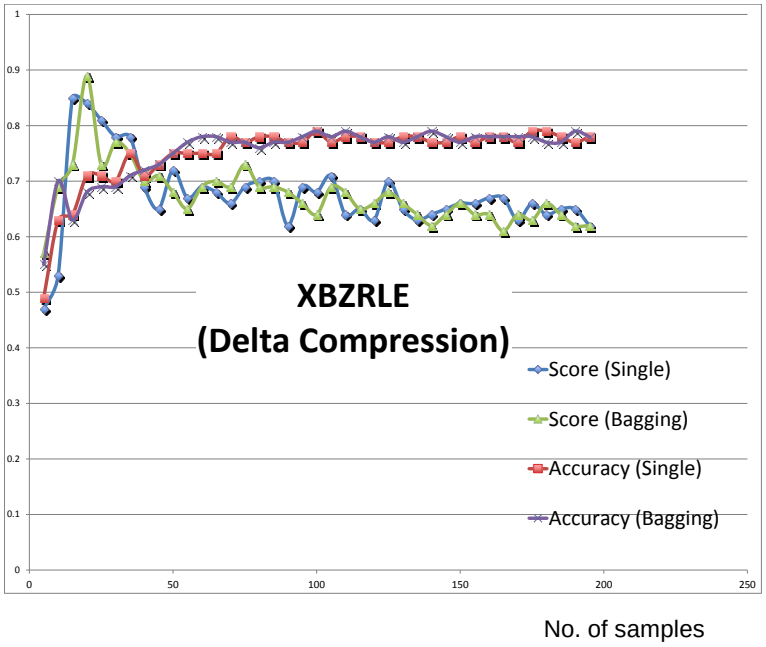



Figure 5.3 Training score and prediction accuracy for XBZRLE

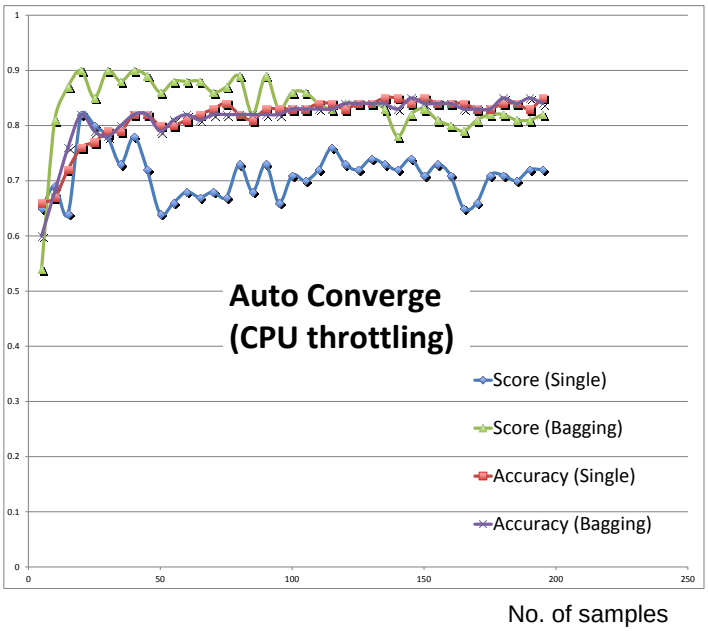



Figure 5.4 Training score and prediction accuracy for compression algorithm

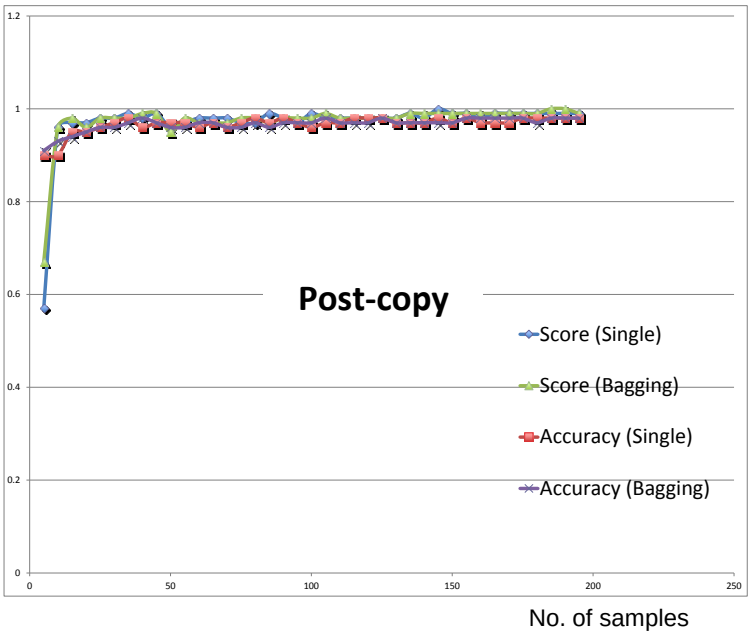



Figure 5.5 Training score and prediction accuracy for post copy algorithm

The number of samples for each predictor started from 10 with pre-defined step value 5 to find threshold values in bootstrapping with multiple predictors. Optional values for kernel function were chosen with static values because they are not affecting results much across the whole set of samples. Thus, C for 1.0, epsilon for 0.1 and gamma for 0.0 were used uniformly respectively.

Table 5.1 – 5.5 shows chosen training parameters and predicted total migration time from absolute error on average $90^{th}$ for each live migration algorithm on QEMU. In particular, errors in XBZRLE was lower than others, which means delta compression is more predictable than others.

Table 5.1 Parameters and error rate of original pre-copy. avg_dirty_rate : Average Dirty Rate, vm_size : VM's memory size, ws_size : WWS, avg_IPC : Average IPC, L2$_WB_cnt : L2$ write-back count, avg_non_halted_cycles : Average unhalted cycles, avg_wr_density : Average scaled write density

| **Target Metric** | **90th_abs err** | **Training parameters** |
|---|---|---|
| Total migration time | 43.06 | avg_dirty_rate_20sec |
| | 22.10 | avg_dirty_rate_20sec+vm_size |
| | 19.04 | avg_dirty_rate_20sec+vm_size+ws_size |
| | **17.34** | vm_size+avg_dirty_rate_20sec+ws_pages+avg_IPC_20sec |

Table 5.2 Parameters and error rate of original auto-converge

| **Target Metric** | **90th_abs _err** | **Training parameters** |
|---|---|---|
| Total migration time | 46.38 | avg_dirty_rate_20sec |
| | 21.85 | avg_dirty_rate_20sec+vm_size |
| | 16.04 | avg_dirty_rate_20sec+vm_size+ws_size_20sec |
| | **15.30** | vm_size+avg_dirty_rate_20sec+ws_pages+avg_L2$_WB_cnt_20sec+avg_non_halted_cycles |

Table 5.3 Parameters and error rate of original XBZRLE

| **Target Metric** | **90th_abs _err** | **Training parameters** |
|---|---|---|
| Total migration time | 38.73 | avg_dirty_rate_20sec |
| | 14.89 | avg_dirty_rate_20sec+vm_size |
| | 15.44 | avg_dirty_rate_20sec+vm_size+ws_size_20sec |
| | **11.23** | avg_dirty_rate_20sec+avg_wr_density_20sec+ws_pages+non_ws_pages+ws_entropy+non_ws_entropy |

Table 5.4 Parameters and error rate of original compression

| Target Metric | 90th_abs_err | Training parameters |
|---|---|---|
| Total migration time | 39.20 | avg_dirty_rate_20sec |
| | 24.07 | avg_dirty_rate_20sec+vm_size |
| | 19.39 | avg_dirty_rate_20sec+vm_size+ws_size_20sec |
| | **15.76** | avg_dirty_rate_20sec+ws_entropy_pages+non_ws_entropy_pages+ws_entropy+non_ws_entropy |

Table 5.5 Parameters and error rate of original x-postcopy

| Target Metric | 90th_abs_err | Training parameters |
|---|---|---|
| Total migration time | 21.21 | avg_dirty_rate_20sec |
| | 2.60 | avg_dirty_rate_20sec+vm_size |
| | 2.16 | avg_dirty_rate_20sec+vm_size+ws_size_20sec |
| | **2.73** | vm_size |

Table 5.6 shows relative and absolute error on average, 25th, 50th, 90th. We could find the accuracy of fluidanimate Parsec improved much which has extremely low write density. For post-copy, 90th relative error is only 5%. Predicting performance of post-copy seems trivial.

The relative error of downtime is much worse than that of the total migration time. The average relative error of downtime is more than 20% on average for all algorithms except for post-copy. But actually it is not that bad. QEMU tries to guarantee the required downtime 300ms on default, so little absolute error in downtime is exaggerated in relative error. Table 5.6 shows average of 90th absolute error in downtime is only 322ms. Improvement in delta-compression and data-

compression prove the importance of the proposed feature write density and working set and non working set entropy.

Table 5.6 Model accuracy

| | | Relative Error | | | | Absolute Error | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Target Metric** | **Capability** | **avg** | **25th** | **50th** | **90th** | **avg** | **25th** | **50th** | **90th** |
| Total Migration Time | pre-copy | 0.10 | 0.03 | 0.06 | 0.23 | 7.53 | 0.83 | 2.05 | 17.54 |
| | cpu-throttling | 0.09 | 0.03 | 0.06 | 0.21 | 6.88 | 0.76 | 1.86 | 15.43 |
| | delta-compression | 0.09 | 0.03 | 0.06 | 0.20 | 7.61 | 0.69 | 1.79 | 12.77 |
| | data-compression | 0.10 | 0.03 | 0.07 | 0.25 | 5.86 | 1.47 | 2.98 | 14.26 |
| | post-copy | 0.02 | 0.01 | 0.02 | 0.05 | 0.73 | 0.13 | 0.33 | 1.92 |
| Downtime | pre-copy | 0.22 | 0.06 | 0.15 | 0.51 | 138.38 | 35.76 | 76.04 | 322.42 |
| | cpu-throttling | 0.22 | 0.06 | 0.15 | 0.52 | 116.53 | 28.53 | 59.34 | 283.60 |
| | delta-compression | 0.27 | 0.09 | 0.20 | 0.61 | 106.31 | 23.00 | 52.47 | 286.78 |
| | data-compression | 0.37 | 0.07 | 0.22 | 0.75 | 301.32 | 77.41 | 160.61 | 706.43 |
| | post-copy | 0.11 | 0.03 | 0.08 | 0.19 | 27.24 | 3.05 | 5.60 | 39.01 |
| Total Transferred Data | pre-copy | 0.09 | 0.03 | 0.06 | 0.21 | 239.33 | 58.44 | 113.84 | 563.79 |
| | cpu-throttling | 0.09 | 0.03 | 0.06 | 0.21 | 224.26 | 53.63 | 106.83 | 549.28 |
| | delta-compression | 0.10 | 0.03 | 0.07 | 0.21 | 223.69 | 51.75 | 98.49 | 455.70 |
| | data-compression | 0.07 | 0.02 | 0.05 | 0.17 | 60.76 | 18.53 | 35.73 | 128.61 |
| | post-copy | 0.02 | 0.01 | 0.02 | 0.05 | 23.92 | 9.49 | 21.01 | 44.06 |

The total migration time has a high correlation with the total transferred data. For example, the total migration time can be simply approximated by multiplying the available network bandwidth by the total transferred data. Both total migration time and total transferred data graph shows similar trends. The average relative error in the total data transferred is 7.5%.

We tried to predict OLTP performance from migration start to 10sec after using the same features in other metrics. The OLTP performance model failed to predict degradation in performance for pre-copy based algorithms 90th absolute error nearly 60%. The reason is, average of throughput is not much dropped in pre-copy based algorithm and the variance of the throughput is very high. For post-copy, 90th absolute error is 22.6%. It is not that bad number considering the high variance in OLTP performance.

# Chapter 6

# Conclusion

This research shows the potential of a data driven approach to predict live migration performance. The model can be built with higher transparency and extended to parameters which have not been identified yet for various realistic working environments. In addition, being able to build a model with minimum set of samples means that it enables automated migration for real-time migration cases.

As future work, other ensemble methods such as boosting and random forest will be used to build models for predicting the live migration performance, especially where not much training data is available.

# Bibliography


[1] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, “Predicting the Performance of Virtual Machine Migration,” 2010, pp. 37–46.

[2] D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, “Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation,” vol. 5931, M. G. Jaatun, G. Zhao, and C. Rong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 254–265.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live migration of virtual machines,” in *Software, IEEE*, 2005, vol. 21, no. 4, pp. 273–286.

[4] S. Nathan, U. Bellur, and P. Kulkarni, “Towards a Comprehensive Performance Model of Virtual Machine Live Migration,” 2015, pp. 288–301.

[5] L. Deng, H. Jin, H. Chen, and S. Wu, “Migration Cost Aware Mitigating Hot Nodes in the Cloud,” 2013, pp. 197–204.

[6] D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, A. Aldhalaan, and D. A. Menascé, “Analytic Performance Modeling and Optimization of Live VM Migration,” vol. 8168, M. S. Balsamo, W. J. Knottenbelt, and A. Marin, Eds.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 28–42.

[7] D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-Aware Steady State VM Management for Data Centers," vol. 7289, R. Bestak, L. Kencl, L. E. Li, J. Widmer, and H. Yin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 190–204.

[8] Jie Zheng, T. S. E. Ng, K. Sripanidkulchai, and Zhaolei Liu, "Pacer: A Progress Management System for Live Virtual Machine Migration in Cloud Computing," *IEEE Trans. Netw. Serv. Manag.*, vol. 10, no. 4, pp. 369–382, Dec. 2013.

[9] H. Liu and B. He, "VMbuddies: Coordinating Live Migration of Multi-Tier Applications in Cloud Environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 1192–1205, Apr. 2015.

[10] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li, "iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud," *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3012–3025, Dec. 2014.

[11] J. Zhang, F. Ren, and C. Lin, "Delay guaranteed live migration of Virtual Machines," 2014, pp. 574–582.

[12] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, p. 14, Jul. 2009.

[13] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression," 2009, pp. 1–10.

[14] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," *ACM SIGPLAN Not.*, vol. 46, no. 7, p. 111, Jul. 2011.

[15] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-

area migration of virtual machines including local persistent state," 2007, p. 169.

[16] X. Feng, J. Tang, X. Luo, and Y. Jin, "A performance study of live VM migration technologies: VMotion vs XenMotion," 2011, vol. 8310, p. 83101B–83101B–6.

[17] Y. Kuno, K. Nii, and S. Yamaguchi, "A Study on Performance of Processes in Migrating Virtual Machines," 2011, pp. 567–572.

[18] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster Comput.*, vol. 16, no. 2, pp. 249–264, Jun. 2013.

[19] S. Kikuchi and Y. Matsumoto, "Performance Modeling of Concurrent Live Migration Operations in Cloud Computing Systems Using PRISM Probabilistic Model Checker," 2011, pp. 49–56.

[20] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[21] "Wikipedia-Support Vector Machine." [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine.

[22] "Bootstrap Aggregating." [Online]. Available: https://en.wikipedia.org/wiki/Bootstrap_aggregating.

# 요약

가상화는 최근 서버 컴퓨팅 환경이 클라우드로 빠르게 전환되면서 폭넓게 사용되는 기술이다. 시스템 가상화 기능 중 가장 필요한 기술인 라이브 마이그레이션은 활발한 연구 분야였고, 특히 그 성능을 예측하기 위한 시도가 이뤄졌으나 대부분 상대적으로 예측오차가 큰 분석적 방법이거나 또는 실제 환경에서의 추가적인 변수들로 확장하기에 제약이 존재한다. 따라서, 본 논문에서는 기계학습의 한 분야인 서포트 벡터 회귀방식을 기반으로 한 데이터 중심의 접근을 제시한다. 특히 오픈 소스로서 확장성이 뛰어난 시뮬레이터인 QEMU 기반에서 Live Migration의 성능 지표인 전체 마이그레이션 시간, 중단 시간 및 전체 전송량을 예측함으로써, 향후 마이그레이션 자동화를 위한 역할을 제공할 것이다.

**주요어 : virtualization, live migration, machine learning, support vector machine**

**학 번 :** 2007-21003