Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Московский физико-технический институт
(государственный университет)»

Факультет управления и прикладной математики

Кафедра теоретической и прикладной информатики

ОЦЕНКА РАЗМЕРА РАБОЧЕГО НАБОРА ВИРТУАЛЬНОЙ МАШИНЫ ИСХОДЯ ИЗ ГОСТЕВЫХ ПОКАЗАТЕЛЕЙ ПРОИЗВОДИТЕЛЬНОСТИ

Выпускная квалификационная работа
(магистерская диссертация)

Направление подготовки: 01.09.56 Математические и информационные технологии

Выполнил:
студент 973б группы_____ Маркеева Лариса Борисовна

Научный руководитель:
д.ф.-м.н., профессор_____ Тормасов Александр Геннадьевич

Научный консультант:
_____ Мелехова Анна Леонидовна

Москва 2015

Оглавление

ГЛАВА 1

Введение

Цель данной магистерской диссертации разработать алгоритм, который оценивает размер рабочего набора виртуальной машины и принимает решение о том, сколько оперативной памяти можно изъять у конкретной виртуальной машины(virtual machine, VM) для перераспределения между другими VM, запущенными на той же хост-машине (host machine).

Подразумевается, что хост-машина может являтся частью кластера или облака.

Объясним некоторые из понятий:

Виртуальная машина *(virtual machine, VM, гость, guest machine)* - программа эмулирующая некоторую существующую или гипотетическую архитектуру ЭВМ и создана с целью выполнения на ней программ. Виртуальные машины применяются в случае, когда некоторая архитектура недоступна или для оптимизации использования ресурсов хост-машины. В “Formal requirements for virtualizable third generation architectures”[1] виртуальная машина объявленна как “эффективная, изолированная копия реальной машины”.

Хостовая машина *(хост-машина, хост, host machine)* - реальный ЭВМ на котором запущена одна или несколько виртуальных машин. Предоставляет свои физические ресурсы для их нужд.

Гипервизор *(монитор виртуальной машины, монитор, hypervisor, virtual machine monitor, VMM)* - программно-аппаратный комплекс, который создает и обеспечивает работу виртуальной машины на хостовой машине.

Размер рабочего набора *(working set size)* - это количество оперативной памяти, которое необходимо для вычисления поставленной задачи[2]. Для высоконагруженных систем, где ошибка в расчете размера рабочего набора может оказаться очень дорогой, размер рабочего набора объявлен как размер памяти, который обеспечивает необходимую производительность системы. В данной работе используется второе определение размера рабочего набора.

Миграция *(живая миграция, migration)* - перенос виртуальной машины с одного физического сервера(хост-машины) на другой без прекращения его работы.

Переподиска *(oversubscription, overcommitment)* - методы, которые обеспечивают возможность связать с виртуальными машинами данного хоста больше ресурсов, чем доступно на хост-машине[3]. В частности на рисунке 1.1 суммарный объем оперативной памяти всех виртуальных машин больше, чем количество доступной RAM на хост-машине.

Самым безопасным способом распределения ресурсов является метод, когда с виртуальными машинами связывают только ресурсы, которые действительно доступны на хост-машине. В таком случае виртуальная машина всегда будет иметь доступ к выделенным для нее ресурсам. С другой стороны, как показанно в [3], виртуальные машины в среднем используют 5-15% выделенного процессорного времени. Остальное процессорное время можно перераспределить между другими виртуальными машинами, запущенными на данном хосте. Однако, в случае неожиданного увеличения нагрузки на виртуальную машину может произойти потеря производительности или даже миграция на другой хост из-за недостатка ресурсов.

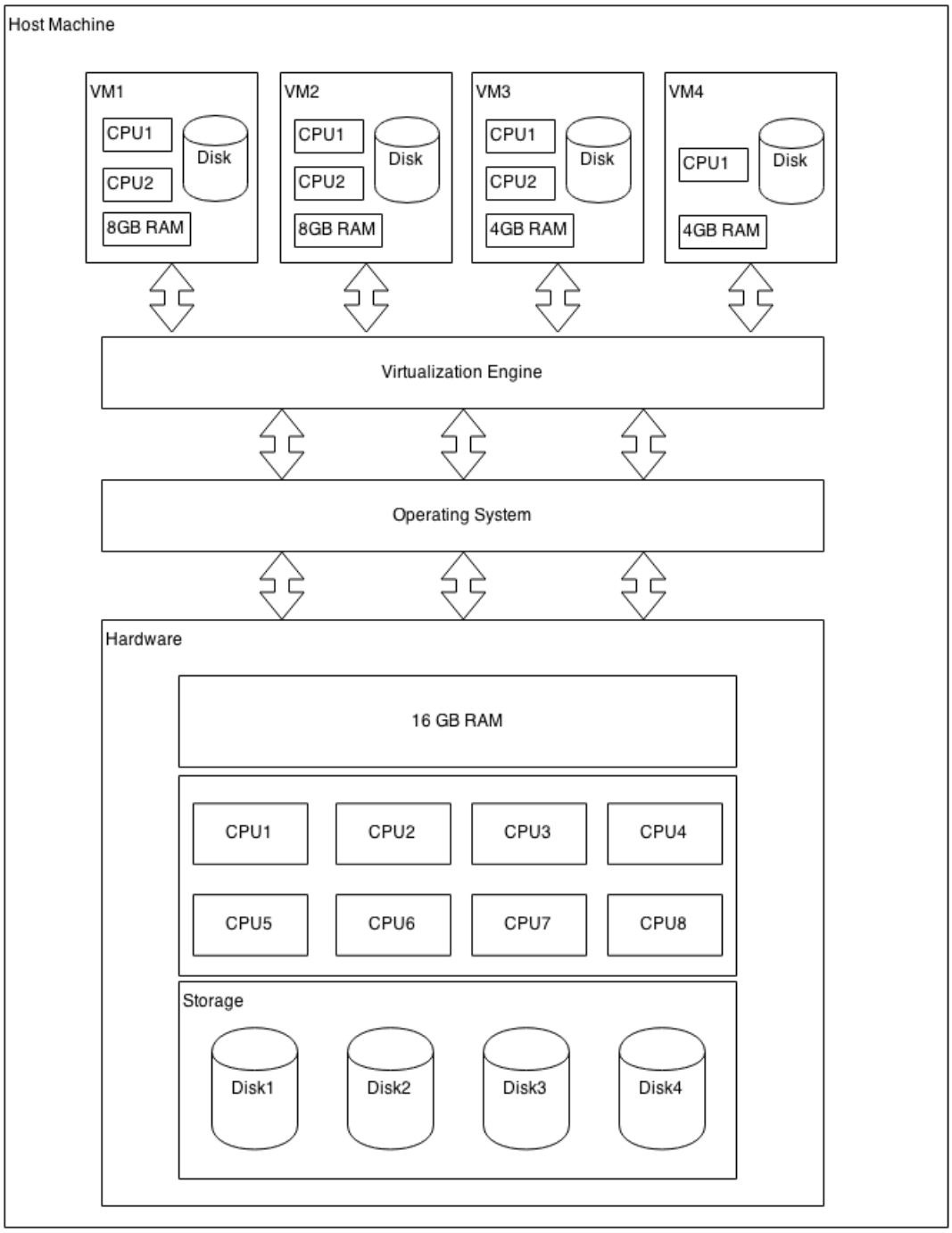

Рис. 1.1: Схема виртуализации.

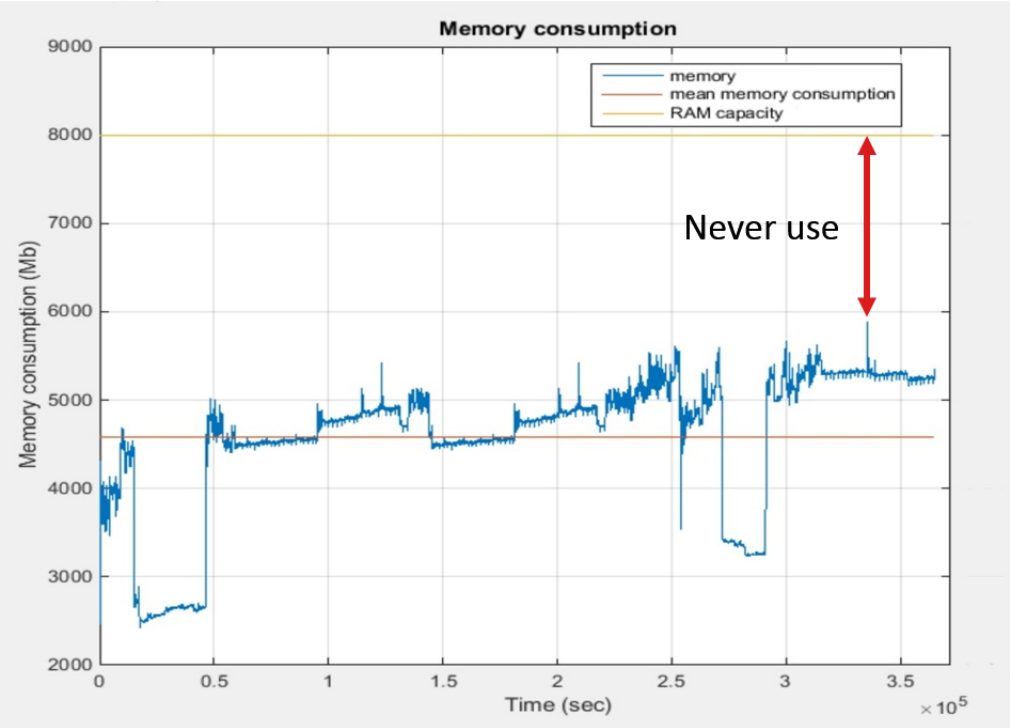

Рис. 1.2: График потребления памяти виртуальной машиной.

Таким образом, переподписка позволяет увеличить эффективность использования ресурсов хост-машины, с другой стороны переподписка может привести к потере производительности.

На рисунке 1.2 представлен график потребления памяти виртуальной машиной. По горизонтали отложенно время, по вертикали количество используемой памяти в мегабайтах. Синяя линия показывает объем потребляемой памяти в каждый момент времени. Красная линия показывает потребление памяти в среднем за все время наблюдений. Оранжевая линия демонстрирует размер RAM виртуальной машины. Как видно из рисунка примерно 2 GB памяти не использовались виртуальной машиной на протяжении всего наблюдения.

Неиспользуемую память можно перераспределить между другими виртуальными машинами, запущенными на хосте.

Задача вычисления размера рабочего набора, который обеспечит необходимый уровень производительности системы для перераспределения оставшейся свободной памяти решается в данной работе.

Алгоритмы менеджмента памяти отличаются в различных ОС, таких как Windows OS, Linux OS. В данной работе особый упор сделан на Windows OS (Windows 7, Windows 8, Windows 8.1), но полученный результат легко может быть расширен и на другие операционные системы.

Количество памяти, необходимой для заданного уровня производительности напрямую зависит от нагрузки, которую испытывает виртуальная машина в данный момент и будет испытывать в будущем. Для измерения степени нагруженности на систему используются ***performance counters***, встроенные в операционную систему

Windows.

Глава 2 проводит краткий обзор литературы по данной проблематике.

Глава 3 описывает теоретические аспекты, которые были применены в данной работе, *performance counters*, свойства временных рядов и их последствия, алгоритмы кластеризации, типы метрик.

Глава 4 описывает используемые в работе технические средства и программные пакеты.

Глава 5 описывает результаты статистического анализа временного ряда размера рабочего набора виртуальной машины.

Глава 6 описывает предложенные в данной работе модели оценки размера рабочего набора виртуальной машины.

Глава 7 описывает результаты тестирования моделей из главы 6.

Глава 8 кратко описывает результаты данной работы и предлагает возможные дальнейшие варианты исследований.

Глава 9 содержит заключение по данной работе.

ГЛАВА 2

Обзор Литературы

В этой главе представлен краткий обзор литературы по проблеме эффективного менеджмента памяти виртуальных машин и оценок размера рабочего набора виртуальных машин.

Эффективный менеджмент ресурсов в облачных системах является очень важной проблемой. В 2006 году дата-центры потребляли 61 миллиард киловатт-часов электроэнергии, что составляло 1.5% потребления электроэнергии в США [4]. В 2014 году потребление электроэнергии выросло до 91 миллиарда киловатт-часов, а в 2020 прогнозируется 139 миллиардов киловатт-часов [5]. При этом рост стоимости электроэнергии составит примерно 4.7$. Поэтому, чем больше виртуальных машин может быть запущенно на одной хост-машине без потери производительности, тем дешевле дата-центры будут обходиться владельцу. Например в [6] рассматривается алгоритм ***Green Scheduling Algorithm*** в кооперации с нейронной сетью. Нейронная сеть предсказывает возможные нагрузки и *Green Scheduling Algorithm* включает или выключает ноды облака в зависимости от нагрузки. Таким образом минимизируется количество запущенных нод в облаке.

Так же помимо таких глобальных методов оптимизации как включение или отключение нодов существуют методы оптимизации отдельных видов ресурсов, таких как CPU, оперативная память, хранилища данных и т.д. Стоит отметить, что CPU и оперативная память являются наиболее активно потребляемыми ресурсами хостовой машины [3, 7].

В [7] описан метод оптимизации потребления CPU, работающего на архитектуре Intel x86. ***CPU idle state*** - состояние CPU при котором для него нет задач. В таком состоянии операционная система посылает команду *halt* процессору для снижения энергопотребления. Эта команда значит, что на данной виртуальной машине в данный момент очень низкая нагрузка на CPU. Монитор виртуальной машины перехватывает это событие, и сообщает системе, отвечающей за перераспределение ресурсов о том, что на данной виртуальной машине низкая нагрузка на CPU, а следовательно, ресурс можно частично изъять и перераспределить между другими виртуальными машинами на этом хосте. К сожалению, столь явного признака низкой нагрузки для оперативной памяти не существует.

В работе [8] отмечается существенное отличие в поведении grid-систем и облачных систем. Основным отличием облачных систем являются более сильные и частые колебания в потреблении памяти, чем у grid-систем.

Для оптимизации использования оперативной памяти используются различные методы, такие как:

1. Page Replacement [8];
2. Ballooning [8, 3, 9];
3. Demand Paging[8];
4. Sharing Memory[8, 3];
5. Reclaim Idle Memory[8];

6. Различные техники предсказания размера рабочего набора виртуальной машины [4, 7, 10, 11, 12];

Page Replacement - идея данного метода заключается в перемещении гипервизором некоторых страниц виртуальной машины в свап-файл[13]. Но как показанно в [8] возникает проблема определения страниц, которые надо отозвать. По-факту какие страницы важны, а какие нет знает только виртуальная машина и эта информация недоступна гипервизору. Более того, в случае сложных политик отзыва страниц могут возникнуть так называемые "аномалии производительности" (performance anomaly) [8]. Эти "аномалии" связанны с тем, что некоторые ОС являются закрытыми(Windows, Mac OS) и алгоритмы менеджмента памяти известны не до конца или с тем, что почти все операционные системы имеют недокументированные функции(Windows, Mac OS, Linux), которые активно используются ядром системы[14, 15]. Так же, как показанно в [8] стремление к изолированности виртуальной машины и эффективности использования памяти конфликтуют друг с другом. При использовании *Page Replacement* может возникнуть проблема *double paging problem*. *double paging problem* заключается в следующем: гипервизор выбирает страницу для загрузки ее обратно в оперативную память. В то же время гостевая операционная система принимает решение выгрузить тут же страницу в свап-файл. Таким образом страница будет загруженна в оперативную память и тут же выгруженна.

Ballooning. В данной технике используется специальный драйвер balloon driver, который загружается в гостевую ОС и имеет интерфейс для общения с монитором виртуальной машины. Когда мо-

нитор хочет изъять память в виртуальной машины, он дает через интерфейс драйверу команду “надуться”. Драйвер начинает выделения памяти внутри гостевой машины используя API запущенной на виртуальной машине ОС. Стоит отметить, что “надутый” balloon driver может спровоцировать внутренних алгоритмов управления памятью в ОС. Т.к. если памяти достаточно, то при “надутый” страницы изымаются из буфера свободных страниц. Если памяти мало, то в ОС запускаются алгоритмы освобождения страниц в оперативной памяти, такие как: запись файлового кэша на диск, свапинг, которые так же могут привести к “аномалиям производительности”.

Demand Paging - случайное перемещение страниц виртуальной машины на диск. Используется чтобы избегать “аномалий производительности”, вызванных запуском внутренних алгоритмов ОС управления памяти.

Sharing Memory - если на одной хостовой машине запущенно несколько копий одной и той же ОС. Очевидно, что большая часть страниц памяти, которая принадлежит ядру операционной системы, у этих двух виртуальных машин совпадет. В таком случае резонно сделать так, чтобы дублирующиеся страницы были удалены и заменены на ссылки тех мест памяти, где эта страница встречается впервые.

Share-Based Allocation - выделяемые виртуальной машине ресурсы пропорциональны уже потребленным. Как показанно в [8] такой принцип плохо работает при больших нагрузках. Для решении этой проблемы существуют алгоритмы, предложенные в [16, 17].

Reclaiming Idle Memory - недостаток *Share-Based Allocation* заключается в том, что он не учитывает насколько активно вирту-

альная машина использует выделенную память. Идея *Reclaiming Idle Memory* заключается в том, чтобы брать в расчет при распределении ресурсов насколько виртуальная машина активно использует выделенную память.

В работах [4, 7, 10, 11, 12] рассматриваются различные методы оценки размера рабочего набора с помощью различных статистических методов.

В работе [7, 10] рассматриваются возможности использования фильтра Калмана[18] и авторегрессии[19] для предсказания размера рабочего набора в следующий момент времени. Фильтр Калмана предполагает наличие шума исключительно с гауссовым распределением[19]. В добавок ко всему с нестационарными рядами ни фильтр Калмана ни авторегрессия не могут работать корректно[20, 19], т.к. нарушается их основное предположение о стационарности ряда (неизменности дисперсии, среднего и ковариации между лагами с течением времени). Так же в работе [10] отмечается, что использование фильтра Калмана для долгосрочных предсказаний крайне неудобно. Т.к. в случае предсказания на 16 часов вперед корректировка настроек фильтра Калмана произойдет толко через 16 часов.

В работе [10] так же перечисляется ряд стандартных моделей для оценки размера рабочего набора:

1. ***Last-state based method*** - следующее значение размера рабочего набора равно текущему;

2. ***Mean-average*** - усредненное значение за последние n моментов времени является предиктором значения размера рабочего набора в следующий момент времени;

3. ***Linear weighted moving average method*** - значение в следующий момент времени равно взвешенной сумме значений размера рабочего набора в предыдущие моменты времени;

4. ***Exponential moving average***:

$$S(t) = \alpha \cdot e_1 + (1 - \alpha) \cdot S(t - 1) \tag{2.1}$$

   Где $S(t)$ - размер рабочего набора в момент времени t, α - эмпирически подобранный параметр;

5. ***Prior probability-based method*** - выбирается наиболее вероятный размер рабочего набора, опираясь на размер рабочего момента в текущий момент времени;

6. ***Auto-regression model***

$$S(t) = \sum_{i=1}^{n} a_i \cdot S(t - i) + \epsilon_t \tag{2.2}$$

   Где $S(t)$ - размер рабочего набора в момент времени t, a_i - коэффициенты при соответствующих лагах, ϵ_t - случайный компонент;

7. ***Hybrid model*** - этот метод использует фильтр Калмана в комбинации с Savitzky-Golay фильтр сглаживания и авторегрессией.

Основной темой [10] описание модели предсказания нагрузок, основанный на байесовском подходе. Суть модели состоит в следующем:

1. Определить вектор интересующих состояний, которые мы хотим предсказывать $W = (w_1, w_2, \cdots, w_m)^T$;

2. Определить вектор признаков от которых зависит предсказываемое состояние. $X = (x_1, x_2, \cdots, x_n)^T$;

3. Расчитать априорные вероятности $P(w_i)$;

4. Расчитать совместную вероятность $P(x_j|w_i), \forall i, j$;

5. Рассчитать апостериорную вероятность по правилу Байеса для каждого w_i из W:

$$P(w_i|x_j) = \frac{P(x_j|w_i) \cdot P(w_i)}{\sum_{k=1}^{m} P(x_j|w_k) \cdot P(w_k)} \tag{2.3}$$

6. Выбрать тот w_i у которого $P(w_i|x_j)$ наибольшее в качестве предсказанного состояния.

После проведения тестирования в [10] было показанно, что байесовский подход является более точным, чем вышеперечисленные методы. Недостатком данного метода является то, что если появляется сильно отличающееся от тренировочной выборки поведение, классификатор начинает сильно ошибаться.

В работе [21] предложен метод предсказания нагрузок на оперативную память при помощи нейронных сетей, позволяющий достичь 48% улучшения эффективности использования оперативной памяти.

В работе [4] показанно что линейная регрессионная модель показывает лучший результат при предсказании нагрузок на память, чем нейронная сеть.

ГЛАВА 3

Теория

В данной главе рассматриваются теоретические аспекты, которые были использованны в данной работе.

В параграфе 3.1 описывается понятие *события* для систем виртуализации. В частности события Windows OS.

В параграфе 3.2 рассматриваются основные моменты теории временных рядов, которая будет применена в последствии для выбора подходящих методов и оценки размера рабочего набора виртуальной машины.

В параграфе 3.3 рассматриваются алгоритмы кластеризации и различные метрики, используемые в данной работе.

3.1 Внутренние события OS (Events)

Идея *событий*(events) активно используется в данной работе. Информация о происходящей событиях на виртуальной машине анализируется для оценки нагрузки на систему в данный момент времени и предсказания будущих нагрузок.

В работах[7, 22, 9, 23] описана идея событий(events). Существует два типа событий:

1. ***Виртуализационные события*** - события в гостевой ОС, для обработки которых необходимо переключение в монитор виртуальной машины. Т.е. операционная система "останавливается", происходит переключение в гипервизор, который обрабатывает событие(например, запрос в PCI шине), после чего

управление возвращается обратно операционной системе гостевой машины.

2. ***Внутренние события*** - события которые обрабатываются самой гостевой ОС без переключения в гипервизор.

Рассмотрим, что представляют собой *виртуализационные события.* Главная задача виртуализации с точки зрения производительности – минимизировать накладные расходы, порождаемые технологией. Чем дольше виртуальный процессор будет работать в режиме непосредственного исполнения, чем меньше переключений в гипервизор будет совершено, тем лучше. Однако, некоторые инструкции и события виртуального процессора не могут быть исполнены без переключения в гипервизор, и поэтому их необходимо виртуализовать (обработать при помощи гипервизора и вернуть результат гостевой системе. Эмулируя тем самым работу с реальным hardware для гостевой ОС). Эти события будем здесь и далее называть виртуализационными событиями[24].

Большая часть виртуализационных событий связана с операциями, исполняемыми ядром ОС. Более того, часть из операций однозначно коррелирует со случившимися событиями. Например, из количества различных значений CR3 регистра можно узнать количество процессов на системе. Действительно, каждый процесс в современной операционной системе имеет отдельное адресное пространство, а значит, свое *страничное преобразование.* В архитектуре Intel x86 корень дерева страничных преобразований хранится в регистре CR3[25]. Соответственно, изменение CR3 говорит о том, что новый процесс был поставлен на исполнение. Другим примером является

регистр IA32_FSBASE_MSR, запись в котором свидетельствует о переключении между kernel и user уровнем в Microsoft Windows x64.

Внутренние события - события, которые обрабатываются самой гостевой ОС без переключения в гипервизор. К таким событиям можно отнести переключения окон в графическом интерфейсе операционной системы, запросы к API операционной системы и т.д. В данной работе мы активно работаем с *performance counters*(счетчики производительности) операционной системы Windows[26].

Performance counters операционной системы Windows OS - это набор счетчиков операционной системы, показывающих как хорошо операционная система, приложение, сервис или драйвер справляются со своей задачей(насколько высока производительность). Эти счетчики помогают оценить нагрузку на систему и найти узкие места(bottleneck) в системе, на которых теряется большая часть производительности[

В таблице 3.1 описанны используемые для предсказания нагрузки *performance counters.*

Имя счетчика	Описание
Use phys memory	Количество используемой оперативной памяти в данный момент времени в страницах.
CommitTotal	Количество используемых страниц в оперативной памяти и свап-файлах.
CommitLimit	Максимальное количество страниц в системе, которое может быть использовано в системе без дополнительных действий. Т.е. без подключения дополнительного свап-файла. Это не жесткий лимит, он может изменяться.

CommitPeak	Максимальное количество страниц которое использовалось системой с момента ее старта.
PhysicalTotal	Размер оперативной памяти в страницах.
PhysicalAvailable	Размер свободной оперативной памяти в страницах. Включает в себя, в том числе системный кеш.
SystemCache	Размер кеша системы в страницах.
KernelNonpaged	Количество невытесняемых (не могут быть выгруженны в свап-файл) пулов операционной системы в страницах.
KernelPaged	Количество вытесняемых пулов операционной системы в страницах.
KernelTotal	Сумма KernelNonpaged и KernelPaged.
PageSize	Размер страница памяти. Обычно 4096 байт.
HandleCount	Количество открытых дескрипторов в данный момент времени.
ProcessCount	Количество запущенных в данный момент процессов.
ThreadCount	Количество запущенных в данный момент потоков.
Number of processors	Количество CPU компьютера.
PageFileSize	Размер свап-файла(файла подкачки).
PageFileUse	Количество используемого пространства в свап-файле в страницах.
PageFaultCount	Количество Page Faults с момента последнего чтения PageFaultCount.

Processors utilization	Степень загруженности процессоров в данный момент времени в процентах.
IoReadTransferCount	Количество байт прочитанных при помощи вызова ZwReadFile, с момента последнего чтения IoReadTransferCount.
IoWriteTransferCount	Количество байт записанных при помощи вызова ZwWriteFile с момента последнего чтения IoWriteTransferCount.
IoOtherTransferCount	Количество байт, которые были использованны в остальных операциях I/O, таких как ZwDeviceIoControlFile с момента последнего чтения IoOtherTransferCount.
IoReadOperationCount	Количество вызовов ZwReadFile, с момента последнего чтения IoReadOperationCount.
IoWriteOperationCount	Количество вызовов ZwWriteFile, с момента последнего чтения IoWriteOperationCount.
IoOtherOperationCount	Количество вызовов других операций I/O, таких как ZwDeviceIoControlFile, с момента последнего чтения IoOtherOperationCount.
SystemCalls	Количество выполненных системных вызовов, с момента последнего чтения SystemCalls.
ContextSwitches	Количество переключений контекста, с момента последнего чтения ContextSwitches.
CopyOnWriteCount	Количество page faults вызванных попыткой записи в copy-on-write память, с момента последнего чтения CopyOnWriteCount.

TransitionCount	Количество soft page faults, с момента последнего чтения TransitionCount.
DemandZeroCount	Количество demand zero faults, с момента последнего чтения DemandZeroCount.
PageReadCount	Количество прочитанных страниц, с момента последнего чтения PageReadCount.
PageReadIoCount	Количество операций чтения, вызванных page fault, с момента последнего чтения PageReadIoCount.
CacheReadCount	Количество операций чтения кеша с диска, вызванных page fault, с момента последнего чтения CacheReadCount.
CacheIoCount	Количество операций чтения кеша, вызванных page fault, с момента последнего чтения CacheIoCount.
PagedPoolAllocs	Объем выделенной памяти из пула операционной системы.
PagedPoolFrees	Количество памяти возвращенной в пул.

Таблица 3.1: Список используемых в работе performance counters.

Некоторые из приведенных в таблице 3.1 счетчиков на прямую указывают на текущую нагрузку на систему, например *Processors utilization*. Так же в главе 5 будет рассмотренно влияние значения счетчиков в данный момент времени на размер рабочего набора в следующий момент времени.

Некоторый из параметров явно зависят друг друга. Например, *Kernel Total* зависит от *KernelNonpaged* и *KernelPaged*. Так же возможна ситуация при которой *performance counters* имеют нетривиальную линейную зависимость.

В секции 3.2.6 рассматривается проблема мультиколлинеарности (нетривиальной линейной зависимости параметров регрессии друг от друга) и способы обхода этой проблемы при помощи метода главных компонент из раздела 3.2.7.

3.2 Анализ временных рядов

3.2.1 Линейная регрессия

Регрессия - зависимость величины X(зависимой переменной) от одной или нескольких переменный $Y = (y_1, y_2, \cdots, y_m)^T$ (регрессор). В отличии от чисто функциональной зависимости $y = f(x)$, когда каждому значению независимой переменной x соответствует одно определенное значение y, при регрессионой связи одному и тому же значению x могут соответствовать в зависимости от случая различные значения величины y[19].

Многомерная ***линейная регрессия*** имеет вид:

$$Y = \beta \cdot X + \epsilon \tag{3.1}$$

Y - вектор значений зависимой переменной в моменты времени от 0 до k.

$\beta = (\beta_0, \beta_1, \beta_2, \cdots \beta_k)$ - вектор коэффициентов при регрессорах.

X - матрица размером $(n \times k)$, матрица наблюдений (строки матрицы — векторы значений факторов в данном наблюдении, по столбцам — вектор значений данного фактора во всех наблюдениях).

ϵ - ошибка модели.

3.2.2 Метод наименьших квадратов (Ordinary Least Squares)

Обычно обучение модели линейной многомерной регрессии выполняется при помощи ***метода наименьших квадратов***(МНК, Ordinary Least Squares, OLS).

Обучение модели состоит в подборе коэффициентов $\hat{\beta} = (\beta_0, \beta_1, \cdots, \beta_k)$. Теория данного метода представленна ниже.

Из уравнения 3.1 следует что, вектор зависимой переменной и вектор ошибки будут иметь зависимость:

$$\hat{Y} = \beta X,\ e = Y - \hat{Y} = Y - \beta X,\ i = 0 \cdots n \tag{3.2}$$

Соответственно сумма сумма квадратов остатков регрессии(residual sum of squares) в матричном виже будет равна:

$$RSS = e^T e = (Y - \beta X)^T (Y - \beta X) \tag{3.3}$$

Дифференцируя эту функцию по вектору параметров β и приравняв производные к нулю, получим систему уравнений (в матричной форме):

$$\left(X^T X\right) b - X^T Y = 0 \tag{3.4}$$

$$\left(X^T X\right) b = X^T Y \tag{3.5}$$

Решеним уравнение 3.5 относительно β:

$$\hat{\beta}_{OLS} = \left(X^T X\right)^{-1} X^T Y \tag{3.6}$$

Для возможности оценки методом OLS должны быть выполненны следующие предпосылки:

1. Истинная зависимость имеет вид:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \tag{3.7}$$

2. Наблюдений больше чем оцениваемых коэффициентов β:

$$n > k \tag{3.8}$$

3. Строгая экзогенность:

$$E\left(\epsilon_i | X\right) = 0 \tag{3.9}$$

4. Условная гомоскедастичность:

$$E\left(\epsilon_i^2 | X\right) = \sigma^2 \tag{3.10}$$

5. Условная некоррелированность:

$$Cov\left(\epsilon_i \epsilon_j | X\right) = 0, i \neq j \tag{3.11}$$

6. Нет линейной зависимости между регрессорами

7. Векторы отдельных наблюдений независимы и одинаково распределенны

Как показано в [19] оценки коэффициентов методом OLS обладают целым рядом полезных свойств:

1. ***Базовые свойства*** - свойства верные даже без предположении о нормальности распределения ϵ_i и не зависят от размера выборки.

   (a) Оценки β_j линейны по Y:

   $$\beta_j = c_0 y_0 + c_1 y_1 + \cdots + c_n y_n \tag{3.12}$$

   (b) Оценки несмещены условно:

   $$E\left(\hat{\beta}_j | X\right) = \beta_j \tag{3.13}$$

   (c) Оценки несмещены безусловно:

   $$E\left(\hat{\beta}_j\right) = \beta_j \tag{3.14}$$

   (d) Оценки являются несмещенными среди линейных и несмещенных оценок, т.е. для любой линейной по y_i и несмещенной альтернативной оценки $\hat{\beta}^{alt}$:

   $$Var\left(\hat{\beta}_j^{alt} | X\right) \geq Var\left(\hat{\beta}_j | X\right) \tag{3.15}$$

   $$Var\left(\hat{\beta}_j^{alt}\right) \geq Var\left(\hat{\beta}_j\right) \tag{3.16}$$

   (e)

   $$Var\left(\hat{\beta}_j | X\right) = \frac{\sigma^2}{RSS_j} \tag{3.17}$$

(f)

$$Cov\left(\hat{\beta}_j, \hat{\epsilon}_j | X\right) = 0 \tag{3.18}$$

(g)

$$E\left(\hat{\sigma}^2 | X\right) = \sigma^2 \tag{3.19}$$

$$E\left(\hat{\sigma}^2\right) = \sigma^2 \tag{3.20}$$

$$\hat{\sigma}^2 = \frac{RSS}{n-k} \tag{3.21}$$

2. ***Асимптотические*** - свойства которые верны на больших выборках даже без предположении о нормальности распределения ϵ_i. При $n \to \infty$:

   (a)

   $$\hat{\beta}_j \to \beta_j \tag{3.22}$$

   по вероятности, т.е. $\hat{\beta}_j$ состоятельны

   (b)

   $$\frac{\hat{\beta}_j - \beta_j}{se\left(\hat{\beta}_j\right)} \sim N(0,1) \tag{3.23}$$

   (c) Сходимость по вероятности

   $$\hat{\sigma}^2 \to \sigma^2 \tag{3.24}$$

   $$\hat{\sigma}^2 = \frac{RSS}{n-k} \tag{3.25}$$

3. ***При нормальности ошибок*** - свойства верные даже на малых выборках при нормально распределенных ϵ_i. $\epsilon_i|X \sim N\left(0, \sigma^2\right)$:

   (a) Оценки эффективны среди несмещенных

   (b)
$$\left.\frac{\hat{\beta}_j - \beta_j}{se\left(\beta_j\right)}\right| X \sim t_{n-k} \tag{3.26}$$

$$\frac{\hat{\beta}_j - \beta_j}{se\left(\beta_j\right)} \sim t_{n-k} \tag{3.27}$$

   (c)
$$\left.\frac{RSS}{\sigma^2}\right| X \sim \chi^2_{n-k} \tag{3.28}$$

$$\frac{RSS}{\sigma^2} \sim \chi^2_{n-k} \tag{3.29}$$

3.2.3 Оценка качества моделей

В данном разделе рассматриваются методы оценки качества полученных моделей.

Существует три ключевые метрики оценки модели из которых выводятся все остальные:

1. ***Residuals sum of squares***:
$$RSS = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 \tag{3.30}$$

   Где y_i - истинное значение зависимой переменной в момент времени i, $\hat{y}_i$ - предсказанное полученной моделью значение.

2. ***Explained sum of squares***:

$$ESS = \sum_{i=1}^{n} (\hat{y}_i - \overline{y})^2 \tag{3.31}$$

Где $\hat{y}_i$ - предсказанное полученной моделью значение, $overliney$ - среднее значение $\overline{y}$ - среднее значение по набору данных, по которому подгонялась модель.

3. ***Total sum of squares***:

$$TSS = \sum_{i=1}^{n} (y_i - \overline{y})^2 \tag{3.32}$$

Где y_i - истинное значение зависимой переменной в момент времени i, $\overline{y}$ - среднее значение зависимой переменной.

Эти метрики пригодны для сравнения одной и той же модели с различными параметрами.

Для сравнения моделей используются следующие метрики:

1. ***R squared***:

$$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS} \tag{3.33}$$

2. ***R squared adjusted***:

$$R^2_{adj} = 1 - \frac{\frac{RSS}{n-k}}{\frac{TSS}{n-1}} = 1 - \left(1 - R^2\right) \frac{n-1}{n-k} \tag{3.34}$$

3. ***Информационный Критерий Акаике (AIC)***:

$$AIC = \frac{2k}{n} + \ln\left(\frac{ESS}{n}\right) \tag{3.35}$$

4. ***Байесовский Информационный Критерий Шварца (BIC)***:

$$BIC = \frac{k \ln(n)}{n} + \ln\left(\frac{ESS}{n}\right) \tag{3.36}$$

Основная проблема применения (выборочного) R^2 заключается в том, что его значение увеличивается (не уменьшается) от добавления в модель новых переменных, даже если эти переменные никакого отношения к объясняемой переменной не имею, поэтому сравнение моделей с разным количеством факторов с помощью коэффициента детерминации, вообще говоря, некорректно. Для сравнения моделей с разным количеством факторов используют критерии R^2_{adj}, AIC,BIC.

При сравнении моделей лучшей является та модель у которой R^2_{adj} больше или AIC, BIC меньше.

3.2.4 Проверка гипотезы о незначимости регрессии

Так же помимо критериев для сравнений моделей существует критерий для проверки значимости регрессии. Т.е. если наша модель имеет вид:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \tag{3.37}$$

То мы хотим проверить гипотезу, что ни один из регрессоров не

влияет на зависимую переменную:

$$H_0 = \begin{cases} \beta_1 = 0, \\ \beta_2 = 0, \\ \quad \vdots \\ \beta_k = 0. \end{cases} \tag{3.38}$$

Против гипотезы H_1 что хотя бы один из коэффициентов не равен 0.

То для этого используется F-статистика, вычисляемая по формуле:

$$F = \frac{\frac{ESS}{k-1}}{\frac{RSS}{n-k}} \sim F_{k-1,n-k} \tag{3.39}$$

Где n - количество наблюдений, k - количество регрессоров. Если наблюдаемая F-статистика будет больше критического значения, H_0 отвергается.

3.2.5 Тест Рамсея, проверка наличия пропущенных переменных

В своей работе [27] Рамсей предложил тест, проверяющий наличие пропущенных переменных в модели.

Строится дополнительная модель:

$$y_i = x_i^T \beta + a_2 \hat{y}_i^2 + a_3 \hat{y}_i^3 + \cdots + a_m \hat{y}_i^m + \epsilon_i \tag{3.40}$$

И при помощи F-статистики проверяется гипотеза $H_0 : \ a_2 = a_3 = \cdots = a_m = 0$

F-статистика имеет асимптотическое распределение:

$$\chi^2 = \frac{RSS_R - RSS_{UR}}{\frac{RSS_{UR}}{n - k_{UR}}} \to \chi^2_r \tag{3.41}$$

По формуле при нормальном распределении ошибок:

$$F = \frac{(RSS_R - RSS_{UR})/r}{RSS_{UR}/(n - k_{UR})} \sim F_{r,n-k_{UR}} \tag{3.42}$$

RSS_{UR} - RSS модели описанной формулой 3.40.

RSS_R - RSS модели при зануленных коэффициентов из H_0.

r - количество ограничений в H_0.

k_{UR} - количество регрессоров в неограниченной модели.

Если F-статистика больше своего критического значения, гипотеза об отсутствии пропущенных переменных отвергается.

3.2.6 Мультиколлинеаность

Мультиколлинеарность - наличие линейной зависимости между регрессорами[28].

Различают два типа регрессоров:

1. ***Строгая*** - идеальная линейная зависимость между регрессорами;

2. ***Нестрогая*** - зависимость близкая к линейной.

Последствием строгой мультиколлинеарности является не единственность оценок OLS.

Нестрогая мультиколлинеарность не нарушает стандартных предпосылок. Оценки $\hat{\beta}_j$ несмещенные, асимптотически нормально распределенные, можно проверять гипотезы и строить доверительный интервалы. Последствием нестрогой мультиколлинеарности являются высокие стандартные ошибки $se\left(\hat{\beta}_j\right)$.

Последствия нестрогой мультиколлинеарности:

1. Очень широкие доверительные интервалы
2. Незначимые коэффициенты
3. Чувствительность модели к добавлению/удалению наблюдения

Типичные проявления нестрогой мультиколлинеарности:

1. Несколько коэффициентов незначимы по отдельности, но гипотеза об их одновременной незначимости отвергается

Методы борьбы с мультиколлинеарностью:

1. LASSO - регрессия
2. Ridge - регрессия
3. Введение бинарных переменных
4. Метод главных компонент (PCA)

LASSO и Ridge регресси - это регресси со штрафами за слишком большие коэффициенты. Различаются они частью отвечающий за вычисления штрафа.

Бинарные переменные(дамми-переменные) - переменные, которые могут принимать всего два значения, например 0 или 1 в зависимости от признака. Например в модель включен пол человека и дамми-переменная равна 1 в случае, если человек женщина и 0 в случае мужчины.

Метод главных компонент строит новые регрессоры как линейную комбинацию исходных. Подробнее рассмотрен в 3.2.7.

3.2.7 Метод главных компонент (PCA)

Метод главных компонент (*Principal Component Analysis*, *PCA*) — один из основных способов уменьшить размерность данных, потеряв наименьшее количество информации[29]. Фактически, позволяет уменьшить число переменных, выбирая самые изменчивые.

Для понимания рассмотрим простой пример. Пусть у нас есть две исходные переменные $x_1,\ x_2$. PCA сконструирует новые переменные:

$$pc_1 = \frac{1}{\sqrt{2}}x_1 + \frac{1}{\sqrt{2}}x_2,\ pc_2 = \frac{1}{2}x_1 + \frac{\sqrt{3}}{2}x_2 \tag{3.43}$$

При чем pc_1 имеет максимальную выборочную дисперсию, pc_2 имеет вторую по максимальности выборочную дисперсию. При чем pc_1 и pc_2 некоррелированны, следовательно, PCA может быть использован для избежания мультиколлинеарности.

3.2.8 Нормировка

Для того чтобы метод PCA работал нормально, необходимо отнормировать данные.

А данной работе мы использовали обычную Z-нормировку, выполняющуюся по формуле:

$$x_{i_{normalize}} = \frac{x_i - \overline{x}}{se\left(x\right)} \tag{3.44}$$

Где $\overline{x}$ - это математическое ожидание нормируемой последовательности, $se\left(x\right)$ - стандартная ошибка x.

3.2.9 Гомескедастичность и гетероскедастичность

Условная гомоскедастичность:

$$Var\left(\epsilon_i|X\right) = E\left(\epsilon_i^2|X\right) = \sigma^2 \tag{3.45}$$

Условная гетероскедастичность

$$Var\left(\epsilon_i|X\right) = E\left(\epsilon_i^2|X\right) \neq const \tag{3.46}$$

Условная гомоскедастичность:

$$Var\left(\epsilon_i\right) = E\left(\epsilon_i^2\right) = \sigma^2 \tag{3.47}$$

Условная гетероскедастичность:

$$Var\left(\epsilon_i\right) = E\left(\epsilon_i^2\right) \neq \sigma^2 \tag{3.48}$$

Последствия условной гетероскедастичности на малых выборках без предположения о нормальности ϵ_i:

1. Свойство линейности по y сохраняется;

2. Свойство несмещенности оценок:

$$E\left(\hat{\beta}|X\right) = \beta,\ E\left(\hat{\beta}\right) = \beta \tag{3.49}$$

3. Свойство эффективности среди линейных несмещенных оценок не сохраняется.

Последствия условной гетероскедастичности на малых выборках с предположением о нормальности ϵ_i:

1. Свойство:

$$\left.\frac{\hat{\beta}_j - \beta_j}{se\left(\beta_j\right)}\right| X \sim t_{n-k} \tag{3.50}$$

не выполняется, следовательно, больше нельзя строить доверительные интервалы и проверять гипотезы о значимости отдельного коэффициента;

2. Свойство:

$$\left.\frac{RSS}{\sigma^2}\right| X \sim \chi^2_{n-k} \tag{3.51}$$

не выполняется;

3. Свойство:

$$F = \frac{\left(RSS_R - RSS_{UR}\right)/r}{RSS_{UR}/\left(n - k_{UR}\right)} \sim F_{r,n-k_{UR}} \tag{3.52}$$

Снова не выполняется, следовательно, мы больше не можем делать тесты на значимость регрессии.

Последствия условной гетероскедастичности на выборках в которых $n \to \infty$(Асимптотические выборки):

1. Свойство:
$$\hat{\beta} \to \beta \tag{3.53}$$
выполняется;

2. Свойство:
$$\frac{RSS}{n-k} \to \sigma^2 \tag{3.54}$$
выполняется;

3. Свойство:
$$\frac{\hat{\beta}_j - \beta_j}{se\left(\beta_j\right)} \sim N\left(0,1\right) \tag{3.55}$$
не выполняется, не можем тестировать гипотезы и строить доверительные интервалы нельзя;

4. Свойство:
$$\chi^2 = \frac{RSS_R - RSS_{UR}}{\frac{RSS_{UR}}{n-k_{UR}}} \to \chi^2_r \tag{3.56}$$
не выполняется, проверять гипотезы с несколькими ограничениями нельзя.

Для проверки на условную гетероскедастичность используется два теста: тест Уайта[30] и тест Голдфельда-Квандта[31].

3.2.10 Автокорреляция

Автокорреляция - статистическая взаимосвязь между последовательностями величин одного ряда, взятыми со сдвигом, например, для случайного процесса — со сдвигом по времени. Наличие автокорреляции случайных ошибок регрессионной модели приводит к ухудшению качества МНК-оценок параметров регрессии, а также к завышению тестовых статистик, по которым проверяется качество модели (то есть создается искусственное улучшение качества модели относительно её действительного уровня точности). Поэтому тестирование автокорреляции случайных ошибок является необходимой процедурой построения регрессионной модели.

При автокорреляции нарушается основная предпосылка теоремы Гаусса-Маркова о независимости ошибок модели между собой:

$$Cov\left(\epsilon_i \epsilon_j | X\right) \neq 0, i \neq j \tag{3.57}$$

Например, автокорреляция порядка $p = m$ говорит нам:

$$\epsilon_t = \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \cdots \phi_m \epsilon_{t-m} + u_t \tag{3.58}$$

Где ϵ_t - ошибка в момент времени t, $\epsilon_t - 1$ - ошибка в момент времени $t - 1$ и т.д.

ϕ_i - коэффициены линейной зависимости.

u_t - случайная величина, одинаково распределенная, независимая от регрессоров, $E\left(u_t\right) = 0$, $Var\left(u_t\right) = \sigma_u^2$.

Нарушаемые автокорреляцие предпосылки:

1. Нарушается предпосылка о независимости наблюдений;

2. Часто нарушается предпосылка о строгой экзогенности, например, использоание y_{t-1} нарушает предпосылку $E\left(\epsilon_t|X\right) = 0$.

Последствия автокорреляции на малых выборках без предположения о нормальности ϵ_i:

1. Свойство линейности по y сохраняется;

2. Свойство несмещенности оценок;

$$E\left(\hat{\beta}|X\right) = \beta,\ E\left(\hat{\beta}\right) = \beta \tag{3.59}$$

3. Свойство эффективности среди линейных несмещенных оценок не сохраняется.

Последствия автокорреляции на малых выборках с предположением о нормальности ϵ_i:

1. Свойство:

$$\left.\frac{\hat{\beta}_j - \beta_j}{se\left(\beta_j\right)}\right| X \sim t_{n-k} \tag{3.60}$$

не выполняется, следовательно, больше нельзя строить доверительные интервалы и проверять гипотезы о значимости отдельного коэффициента;

2. Свойство:

$$\left.\frac{RSS}{\sigma^2}\right| X \sim \chi^2_{n-k} \tag{3.61}$$

не выполняется;

3. Свойство:

$$F = \frac{(RSS_R - RSS_{UR})/r}{RSS_{UR}/(n - k_{UR})} \sim F_{r,n-k_{UR}} \tag{3.62}$$

Снова не выполняется, следовательно, мы больше не можем делать тесты на значимость регрессии.

Последствия автокорреляции на выборках в которых $n \to \infty$ (Асимптотические выборки):

1. Свойство:

$$\hat{\beta} \to \beta \tag{3.63}$$

выполняется;

2. Свойство:

$$\frac{RSS}{n-k} \to \sigma^2 \tag{3.64}$$

выполняется;

3. Свойство:

$$\frac{\hat{\beta}_j - \beta_j}{se(\beta_j)} \sim N(0,1) \tag{3.65}$$

не выполняется, не можем тестировать гипотезы и строить доверительные интервалы нельзя;

4. Свойство:

$$\chi^2 = \frac{RSS_R - RSS_{UR}}{\frac{RSS_{UR}}{n - k_{UR}}} \to \chi^2_r \tag{3.66}$$

не выполняется, проверять гипотезы с несколькими ограничениями нельзя.

Выходы:

1. Ослабить предпосылки [28];

2. Использовать устойчивые(робастные) стандартные ошибки:

$$\frac{\hat{\beta}_j - \beta_j}{se_{HAC}\left(\beta_j\right)} \to N\left(0,1\right) \tag{3.67}$$

   Следовательно, робастные стандартные ошибки можно использовать для оценки гипотез и построения доверительных интервалов;

3. Использовать метод максимального правдоподобия (MLE), описанный в секции 3.2.12.

Методы обнаружения:

1. Тест Дарбина-Уотса.

   Предназначен для проверки гипотез об автокорреляциях 1-го порядка:

$$\epsilon_t = \phi_1 \epsilon_{t-1} + u_t \tag{3.68}$$

   Данный тест так же требует предположение о нормальности распределенности остатков и соблюдения условия экзогенности(смотри 3.2.11).

   Гипотезы теста:

$$H_0 : \phi_1 = 0 \tag{3.69}$$

$$H_a : \phi_1 \neq 0 \tag{3.70}$$

2. Тест Бройша-Годфри

   Предназначен для проверки гипотез об автокорреляциях порядка p:

$$\epsilon_t = \phi_1\epsilon_{t-1} + \phi_2\epsilon_{t-2} + \cdots + \phi_p\epsilon_{t-p} + u_t \tag{3.71}$$

   Где ϵ_i - ошибка в момент времени i, ϕ_i - степень влияния данной ошибки, u_t - независимый, одинаково распределенный случайный компонент.

   Не требует предположения о нормальности остатков, допускает нарушение условия экзогенности(смотри 3.2.11).

   Гипотезы теста:

$$H_0 : \phi_1 = \phi_2 = \cdots = \phi_k = 0 \tag{3.72}$$

$$H_a : \exists\phi_i \neq 0,\ i = 1 \cdots k \tag{3.73}$$

3.2.11 Эндогенность и экзогенность

Условие экзогенности - если модель представленна в виде:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \epsilon_i \tag{3.74}$$

То $E\left(\epsilon_i|X\right) = 0$.

Это условие обеспечивает асимптотическую сходимость $\hat{\beta} \to \beta$, и несмещенность оценок МНК $E\left(\hat{\beta}|X\right) = \beta$, $E\left(\hat{\beta}\right) = \beta$

Чаще всего экзогенность факторов постулируется при построении модели. Тем не менее, существуют методы, позволяющие проверить эти предположения. Для подтверждения предположения об экзогенности используется тест Хаусмана.

3.2.12 Метод максимального правдоподобия

Метод максимального правдоподобия - это метод оценивания неизвестного параметра путём максимизации функции правдоподобия[19]. Основан на предположении о том, что вся информация о статистической выборке содержится в функции правдоподобия.

Пусть есть выборка $X_1, \cdots, X_n$ из распределения P_θ, где $\theta \in \Theta$ - неизвестные параметры оцениваемой модели. Пусть $L\left(x|\theta\right) : \Theta \to R$ - функция правдоподобия, где $x \in R^n$. Тогда точечная оценка:

$$\hat{\theta}_{MLE} = \hat{\theta}_{MLE} = arg\ max\ L\left(X_1, \cdots, X_n|\theta\right),\ \theta \in \Theta \tag{3.75}$$

называется оценкой максимального правдоподобия θ. Таким образом оценка максимального правдоподобия — это такая оценка, которая максимизирует функцию правдоподобия при фиксированной реализации выборки.

Обычно вместо функции правдоподобия L используют ее логарифм $l = \ln L$, т.к. это облегчает поиск максимума, таким образом:

$$\hat{\theta}_{MLE} = \hat{\theta}_{MLE} = arg\ max\ l\left(X_1, \cdots, X_n|\theta\right),\ \theta \in \Theta \tag{3.76}$$

Для максимизации данной функции решается следующее уравнение для поиска экстремума:

$$g\left(\theta\right) = \frac{\delta l\left(x, \theta\right)}{\delta \theta} = 0 \tag{3.77}$$

Свойства:

1. Оценки состоятельны:

$$\hat{\theta}_{ML} \to \theta,\ n \to \infty \tag{3.78}$$

2. Асимптотически несмещены:

$$E\left(\hat{\theta}_{ML}\right) \to \theta,\ n \to \infty \tag{3.79}$$

3. Асимптотически эффективны: Дисперсия $Var\left(\hat{\theta}_{ML}\right)$ наименьшая среди асимптотически несмещенных оценок

4. Оценки $\hat{\theta}_{ML}$ асимптотически нормально распределенны:

$$\hat{\theta}_{ML} \sim N\left(\theta, I^{-1}\right),\ n >> 0 \tag{3.80}$$

   Где I - информация фишера, $I = -E\left(l''\left(\theta\right)\right)$

Проверка гипотез производится при помощи специального LR-теста(Likelihood Ration, LR). В этом тесте H_0 - система из q уравнений на неизвестные параметры, H_a - хотя бы одной из q условий не выполненно.

$$LR = 2\left(I\left(\hat{\theta}\right) - I\left(\hat{\theta}_{H_0}\right)\right) \sim \chi_q^2 \tag{3.81}$$

3.2.13 Стационарность

Стационарность - свойство процесса не менять свойства с течением времени. Временной ряд называется стационарным если:

1.

$$E\left(y_i\right) = const \tag{3.82}$$

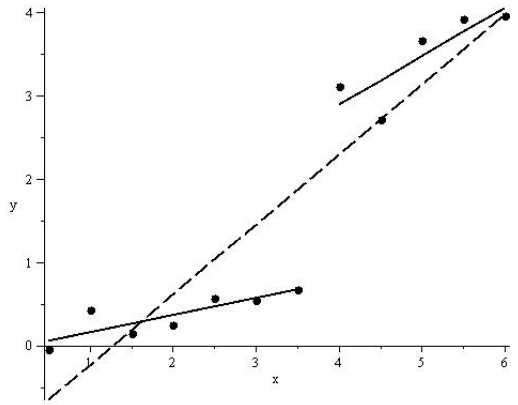

Рис. 3.1: Линейная регрессия с breaks.

2.

$$Var\left(y_i\right) = const \tag{3.83}$$

3.

$$Cov(y_t, y_{t-k}) = \gamma_k \tag{3.84}$$

В случае, если ряд не стационарен[19]:

1. Оценки методом МНК смещенны в сторону 0;

2. t статистика не имеет нормального распределения.

Еще одной проблемой не стохастических рядов являются так называемые *breaks*. *Breaks* - это изменения коэффициентов регрессии с течением времени.

Пример линейной регрессии с breaks приведен на рисунке 3.1. Как видно на промежутке $x = 0 \cdots 3.5$ линейная регрессия имеет одни коэффициенты, после другие.

Следствием такого поведения временного ряда является большие ошибки при предсказании. МНК оценивает регрессию, отображая зависимости "в среднем"[19].

Для определения нестационарности ряда существуют следующие методы:

1. Графический;
2. Тест Дики-Фуллера(Dicky-Fuller test, ADW test);
3. Тест KPSS(Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests).

Так же существуют тесты на детекцию breaks, если их точки не известны, используются: Quandt likelihood ratio(QLR) statistic.

Идея теста заключается в следующем, мы строим новую модель:

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \delta_1 X_1 + \gamma_0 D_t(\tau) + \gamma_1 [D_t(\tau) \times Y_{t-1}] + \gamma_2 [D_t(\tau) \times X_1] + u_t \tag{3.85}$$

Где $D_t(\tau)$ - бинарная переменная, которая равна 0 до момента $t < \tau$, и равна 1 во всех остальных случаях. Тестируется гипотеза: $H_0 : \gamma_0 = \gamma_1 = \gamma_2 = 0$. Для всех τ при помощи F-статистики. Максимальное среди всех проверенных τ значение F-статистики и будет значением Quandt likelihood ratio. Критические значения теста вы можете найти в [19] в соответствующим разделе. Например, для 5% уровня значимости с 6-ю ограничениями(в H_0 входит 6 коэффициентов), это значение равно 3.37.

Как показанно в [19] лучший способ избежать проблем, вызываемых breaks - это изменять параметры регрессии в зависимости от breaks. Способы обнаружения breaks является отдельной темой для исследования.

Так же в [19] указан один способ борьбы с нестационарностью - это "дифференцирование" временного ряда. Т.е. если у нас есть временной ряд $y_t = \{y_0, y_1, \cdots, y_m\}$, то мы строим новый временной ряд вида:

$$\Delta y_i = y_i - y_{i-1},\ i = 1 \cdots m \tag{3.86}$$

К сожалению, как будет показанно в 5 этот метод помогает не всегда.

3.3 Алгоритмы кластеризации

Кластеризация - способ разбития данных на классы по некоторой метрике. Существуют различные виды алгоритмов кластеризации[32]:

1. ***Вероятностные методы***:
   (a) K-средних (K-means);
   (b) K-medians;
   (c) EM-алгоритм;
   (d) Алгоритмы семейства FOREL;
   (e) Дискриминантный анализ;
   (f) DBSCAN.
2. ***Методы искусственного интеллекта***: весьма условная группа, так как методов очень много и методически они весьма различны.
   (a) Алгоритм нечеткой кластеризации C-means;

(b) Нейронная сеть Кохонена;

(c) Генетические алгоритмы.

3. ***Логический методы***. Построение дендрограммы осуществляется с помощью дерева решений.

4. ***Иерархические методы***. Предполагается наличие вложенных групп (кластеров различного порядка). Алгоритмы в свою очередь подразделяются на агломеративные (объединительные) и дивизивные (разделяющие). По количеству признаков иногда выделяют монотетические и политетические методы классификации. Иерархическая дивизивная кластеризация или таксономия.

5. ***Теоретико-графовый методы***.

   (a) Графовые алгоритмы кластеризации

3.3.1 K-Means

Один из наиболее популярных алгоритмов кластеризации. Идея состоит в минимизации суммарного квадратного отклонения объектов от центра кластеров:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_j} (x_j - \mu_j)^2 \tag{3.87}$$

Где k - число кластеров, S_i - полученные кластеры, $i = 1, 2, \cdots, k$ и μ_i - центры масс векторов $x_j \in S_j$.

Алгоритм имеет временную сложность $O\left(2^{\Omega(k)}\right)$[33].

Результат данного метода зависит от начальных установок “центров масс”. Так же существенным недостатком является тот факт, что количество кластеров устанавливает пользователь, а не находится автоматически алгоритмом.

3.3.2 DBSCAN

Алгоритм DBSCAN относится к так называемым density-based[34] алгоритмам. Идея алгоритма заключается в следующем:

1. Объявляется окрестность объекта - r;
2. Объявляется минимальное количество объектов в окрестности - ϵ;
3. Если объект имеет в радиусе r объектов не меньше чем ϵ - это кластер. Все точки находящиеся в радиусе r называются *непосредственно достижимыми*;
4. Если точка, принадлежащая окрестности некоторой точки m(и эта точка входит в кластер) так же в своей окрестности r имеет не мене ϵ соседей, то эта точка и ее соседи в окрестности r присоединяются к кластеру точки m. Такие точки называются просто *достижимыми*;
5. Все недостижимые точки считаются выбросами.

Алгоритм имеет временную сложность $O\left(n \ln n\right)$.

Особенностью алгоритма является то, что в отличии от k-means данный алгоритм возвращает кластеры не сферической формы. Для работы необходимо задать r и ϵ, от которых зависит результат кластеризации.

3.3.3 Иерархическая кластеризация

Существует два направления, называемые сверху-вниз(top-down, нисходящая кластеризация) и снизу-вверх(buttom-up, восходящая кластеризация)[
В первом случае все объекты считаются одним классом, итеративно происходит разбиение класса на подклассы на основе заданной метрики. Восходящий алгоритм действует наоборот. На старте все объекты считаются отдельным классом, алгоритм итеративно объединяет наиболее близкие классы на основе заданной метрики.

Иерархические алгоритмы кластеризации имеют сложность[35]:

1. Восходящие алгоритмы кластеризации имеют сложность $O\left(n^3\right)$;
2. Нисходящие алгоритмы кластеризации имеют сложность $O\left(2^n\right)$.

3.3.4 Метрики

Для использования методов кластеризации необходимо задать функцию метрики, которая будет измерять степень близости объектов между собой. Для сравнения временных рядов могут быть использованны различные метрики:

1. ***Евклидово расстояние*** [34]. Расстояние между точками в n - мерном пространстве в момент времени t рассчитывается по формуле:

$$dist(x_t, y_t) = \sqrt{\sum_{i=0}^{n} (x_{t_i} - y_{t_i})^2} \tag{3.88}$$

Эвклидова расстояние между временными рядами это сумма всех расстояний между точками $t = 0 \dots k$:

$$dist(X, Y) = \sum_{i=0}^{k} (dist(x_i, y_i)) \tag{3.89}$$

2. ***Манхэттенское расстояние*** [34]: Расстояние между точками в n - мерном пространстве в момент времени t рассчитывается по формуле:

$$dist(x_t, y_t) = \sum_{i=0}^{n} |x_{t_i} - y_{t_i}| \tag{3.90}$$

Манхэттенское расстояние между временными рядами это сумма всех расстояний между точками $t = 0 \dots k$:

$$dist(X, Y) = \sum_{i=0}^{k} (dist(x_i, y_i)) \tag{3.91}$$

3. ***Расстояние Чебышева*** [34]: Расстояние между точками в n - мерном пространстве в момент времени t рассчитывается по формуле:

$$l_\infty(x_t, y_t) = max\, |x_{t_i} - y_{t_i}| \tag{3.92}$$

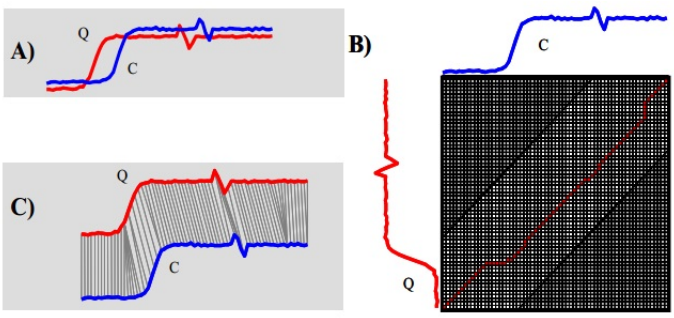

Рис. 3.2: Пример сличения двух одинаковых, но сдвинутых сигналов метрикой Dynamic Time Warping. На картинке A показанны два сравниваемых сигнала, на B показано конструирование матрицы расстояний и оптимальный ее обход(красная линия), C визуализирует сопоставление сигналов.

Расстояние Чебышева между временными рядами это сумма всех расстояний между точками $t = 0 \dots k$:

$$dist(X, Y) = \sum_{i=0}^{k} (dist(x_i, y_i)) \tag{3.93}$$

4. ***Dynamic Time Warping*** (DTW) [36] - специальная метрика, разработанная для сличения звуковых семплов. Не чувствительная к сдвигу, в отличии от перечисленных выше.

   Предположим что имеем два временных ряда:

$$Q = \{q_0, q_1, \dots, q_n\} \tag{3.94}$$

$$C = \{c_0, c_1, \dots, c_m\} \tag{3.95}$$

   На первом этапе вычисления DTW строиться матрица размера $n \times m$, где элемент с номером (i, j) вычисляется по формуле:

$$(i,j) = \sqrt{(q_i - c_j)^2} \tag{3.96}$$

Далее решается задача оптимизации:

$$DTW\,(Q,C) = min\left(\sqrt{\sum_{i=0}^{k} w_k}\right) \tag{3.97}$$

где w_k - это элемент матрицы (i,j), находящейся на k-ом месте в пути W в пути обхода матрицы расстояний. На деле мы пытаемся найти минимальное возможное расстояние с учетом сдвига.

Оптимальный путь может быть найден по рекуррентной формуле:

$$\gamma\,(i,j) = d\,(q_i, c_j) + min\,\{\gamma\,(i-1,j-1)\,,\gamma\,(i-1,j)\,,\gamma\,(i,j-1)\} \tag{3.98}$$

Обход матрицы начинается с правого верхнего угла.

На картинке 3.2 результат сличения двух сигналов.

Алгоритм *Dynamic Time Warping* имеет временную сложность $O\left(n^2\right)$. Но существуют оптимизации, которые позволяют уменьшить временную сложность, такие как *Sakoe-Chiba Band*[37] и *Itakura Parallelogram*[38]. Идея этих методов заключается в ограничении возможности обхода матрица либо двумя линиями, параллельными диагонали или параллелограммом(рисунок 3.3). За пределы этих границ алгоритм поиска оптимального пути выйти не может.

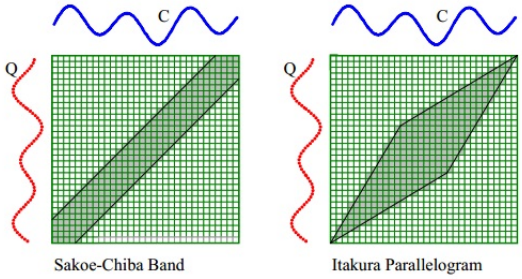

Рис. 3.3: Два наиболее известных способа оптимизации поиска метрики DTW, Sakoe-Chiba Band и Itakura Parallelogram.

ГЛАВА 4

Техническая реализация

В данной главе описываются технические аспекты данной работы.

4.1 Модель взаимодействия

Для экспериментов и разработки алгоритма было использовано следующее ПО:

- Host-machine: Linux Ubuntu 14.04;
- Virtual-machine: Windows 7 или Windows 8;
- Виртуализационное ПО: Parallels Desktop 10;
- VirtIO Ballon driver в качества balloon driver из главы 2;
- Python - для исполнения скриптов управления balloon driver.

Технические характеристики хост-машины:

- Intel Core i5-4570 3.2GHz x 4;
- 16Gb RAM;
- NVidia 770 2GB Video Driver;
- 1Tb HDD.

Технические характеристики виртуальной машины

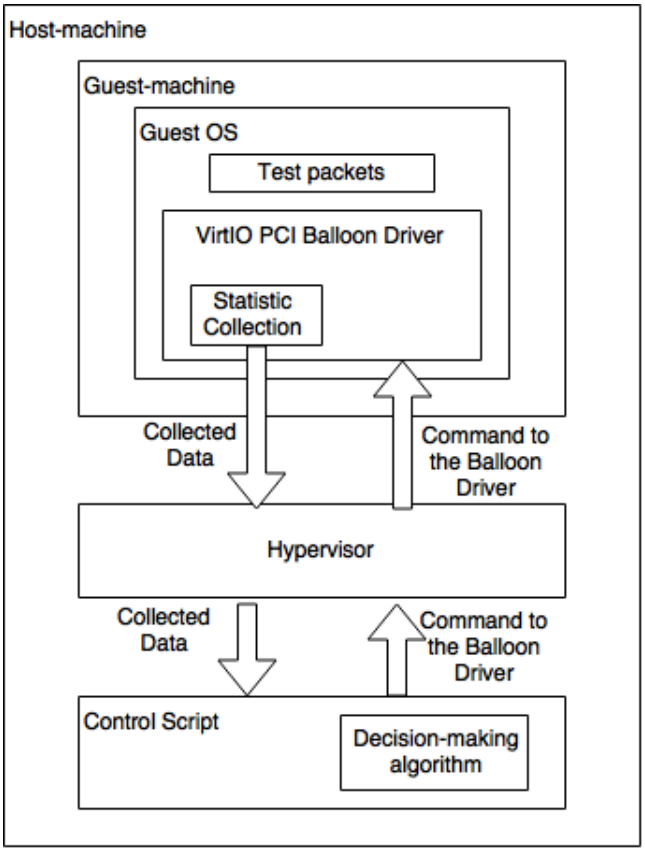

Рис. 4.1: Модель взаимодействия между виртуальной машиной, гипервизором и управляющим скриптом(Control Script).

- 1 CPU;
- 4Gb RAM;
- 256Mb Video Driver;
- 64Gb HDD.

Для сбора статистики специально модифицировался balloon driver. Драйвер получает от ядра гостевой операционной системы счетчики, перечисленные в таблице 3.1. Далее в зависимости от настроек драйвер или передает через специальный интерфейс информацию о состоянии счетчиков гипервизору или сохраняет информацию в файл на самой виртуальной машине. Такой подход позволяет использовать данный драйвер как на виртуальной машине, так и на обычном PC.

Код гипервизора так же был модифицирован. При получении от balloon driver информации о счетчиках операционной системы он записывает файл.

Из файла, в который пишет информацию гипервизор ее вычитывает специальный *Control script*, написанный на языке Python. Данный язык был использован как хорошо адаптированный для прототипирования и имеющий широкие возможности для парсинга файлов. Этот скрипт анализирует полученную информацию и принимает решение о том, сколько памяти в следующий момент balloon driver будет изъято из операционной системы.

Способ анализа и результат принятия решения зависит от модели оценки размера рабочего набора, который в данный момент использует скрипт для расчета размера изымаемой памяти в следующий момент времени. Эти алгоритму будут представленны в главе 6.

После принятия решения через API(Application Program Interface) гипервизора скрипт посылает желаемый размер изъятой памяти balloon driver.

Balloon driver в свою очередь получив новый размер начинает процесс “надутия”(inflation) или “сдутия”(deflation).

Визуализация схемы взаимодействия balloon driver, гипервизора, Control Script представленна на рисунке 4.1.

Так же для тестирования различных способов расчета размера рабочего набора в данной работе использовались atomic tests компании Parallels, PCMark, а так же мануальное тестирование работы виртуальной машины.

4.2 Прочее программное обеспечение

Для модификации balloon driver использовалась IDE eclipse и компилятор gcc.

Для анализа полученной от balloon driver информации использовались следующие математические пакеты: Python, Matlab, R.

ГЛАВА 5

Анализ свойств временного ряда

При помощи системы описанной в 4.1 были собранны данный с 20-ти виртуальных машин. А так же с 15 обычных пользовательских десктопов. Продолжительность наблюдения 24-72 часа.

В данной главе иллюстрацией будет служить конкретный временной ряд, полученные с одной конкретной машины. Но полученные результаты аналогичны и для всех остальных наблюдений(могут немного отличаться значения тестов, но решения о принятии/отвержении гипотезы совпадают).

На этапе подготовки данных они были переформатированны в формат CSV, сохраненны, загруженны в математический пакет R и отнормированы(смотри 3.2.8).

На рисунке 5.1 представлен график изменения потребления памяти в зависимости от времени. На рисунке 5.2 дифференцированный график потребления, вычисленный по формуле 3.86. Этот график показывает на сколько изменилось потребление памяти по сравнению с предыдущим моментов времени. Оба графика построенны по нормированным данным.

5.1 Регрессия Performance Counters

В данном разделе мы попробуем построить линейную регрессию зависимости изменения памяти в следующий момент времени от ***performance counters*** в текущий момент времени, описанных в таблице 3.1. Т.е., фактически, мы пытаемся выяснить, существуют

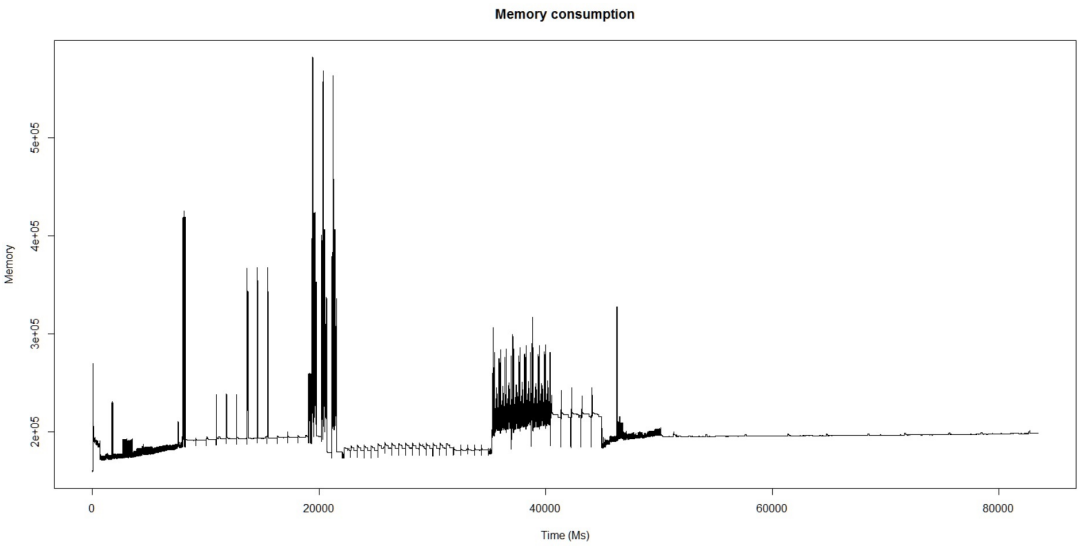

Рис. 5.1: Пример потребления памяти виртуальной машиной.

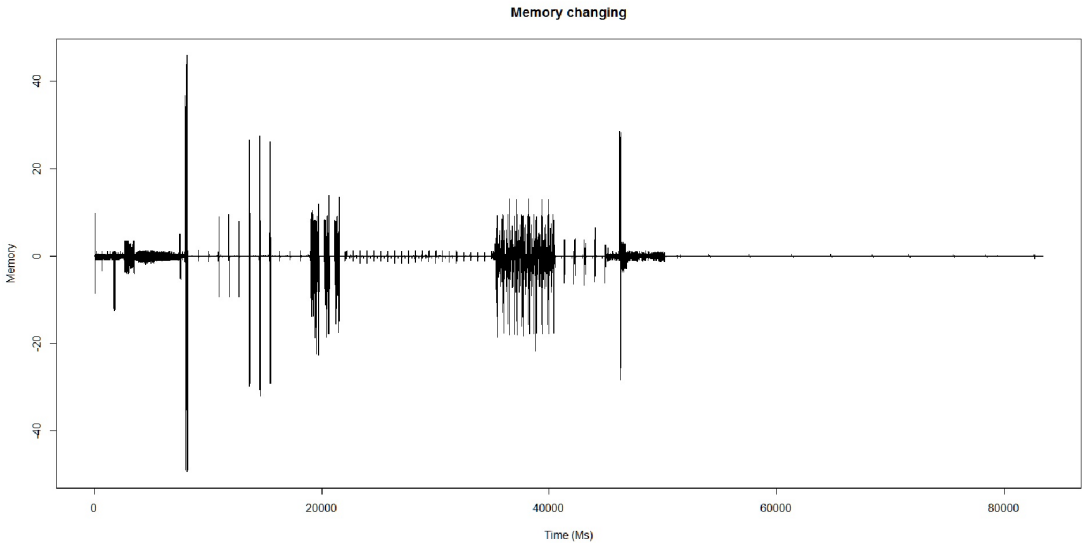

Рис. 5.2: Пример дифференцированного ряда потребления памяти виртуальной машиной.

ли регрессоры которые непосредственно влияют на изменение памяти в следующий момент времени.

Так же на рисунках представленны графики используемых *performance counters*.

Как было замеченно в секции 3.1, используемые счетчики могут иметь между собой нетривиальную линейную зависимость, что может привести к проблеме мультиколлинеарности из раздела 3.2.6. Чтобы избежать этого был применен метод PCA из раздела 3.2.7, для свертки переменных. Результат свертки:

```
## Importance of components:
##                            PC1     PC2     PC3    PC4    PC5     PC6
## Standard deviation     18.4975 1.84647 1.58287 1.4747 1.3717 1.28654
## Proportion of Variance  0.9275 0.00924 0.00679 0.0059 0.0051 0.00449
## Cumulative Proportion   0.9275 0.93675 0.94354 0.9494 0.9545 0.95902
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     1.20364 1.11779 1.06092 1.01437 1.00646 1.00267
## Proportion of Variance 0.00393 0.00339 0.00305 0.00279 0.00275 0.00273
## Cumulative Proportion  0.96295 0.96634 0.96939 0.97218 0.97492 0.97765
##                           PC13   PC14    PC15    PC16    PC17    PC18
## Standard deviation     1.00007 0.9976 0.99443 0.97322 0.92488 0.89670
## Proportion of Variance 0.00271 0.0027 0.00268 0.00257 0.00232 0.00218
## Cumulative Proportion  0.98036 0.9831 0.98574 0.98831 0.99062 0.99280
##                           PC19    PC20    PC21    PC22    PC23    PC24
## Standard deviation     0.85715 0.78077 0.66037 0.48874 0.46613 0.45427
## Proportion of Variance 0.00199 0.00165 0.00118 0.00065 0.00059 0.00056
## Cumulative Proportion  0.99480 0.99645 0.99763 0.99828 0.99887 0.99943
##                           PC25    PC26    PC27      PC28      PC29
## Standard deviation     0.35040 0.22339 0.19760 1.353e-12 1.744e-15
## Proportion of Variance 0.00033 0.00014 0.00011 0.000e+00 0.000e+00
## Cumulative Proportion  0.99976 0.99989 1.00000 1.000e+00 1.000e+00
```

Третья строка в таблице отображает “степень покрытия” дисперсии переменными.Т.е. первая переменная pc_1 покрывает 0.9275, пер-

вая и вторая 0.93675 и т.д. Как видно, 95% дисперсии покрывают первые 5 новых компонент, полученных методом PCA.

Обозначим приращение памяти по сравнению с предыдущим моментов времени за M. Новые переменные как $pc_1, pc_2, pc_3, pc_4, pc_5$. Тогда интересующая нас регрессия примет вид:

$$M = \beta_0 + \beta_1 pc_1 + \beta_2 pc_2 + \beta_3 pc_3 + \beta_4 pc_4 + \beta_5 pc_5 \tag{5.1}$$

Данная регрессия мыла оценена методом наименьших квадратов из раздела 3.2.2 при помощи мат.пакета R:

```
##
## Call:
## lm(formula = memory ~ pca1 + pca2 + pca3 + pca4 + pca5, data = newData2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -49.187  -0.014  -0.012  -0.010  45.931
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.788e-18  3.459e-03   0.000    1.000
## pca1         1.356e-03  1.870e-04   7.251 4.16e-13 ***
## pca2        -2.499e-03  1.874e-03  -1.334    0.182
## pca3         3.137e-03  2.186e-03   1.435    0.151
## pca4         1.433e-02  2.346e-03   6.108 1.02e-09 ***
## pca5         1.226e-02  2.522e-03   4.863 1.16e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9993 on 83444 degrees of freedom
## Multiple R-squared:  0.001405,   Adjusted R-squared:  0.001345
## F-statistic: 23.47 on 5 and 83444 DF,  p-value: < 2.2e-16

## AIC: 236716.5
```

Как мы видим, данная регрессия имеет очень маленький показатель $R^2_{adj} = 0.001345$, что говорит о низкой предсказательной способности данной модели.

В данной работе мы предполагаем, что построенная модель экзогенна.

Проверим основные свойства данного ряда:

- Автокорреляцию, из раздела 3.2.10
- Стационарность, из раздела 3.2.13
- Наличие breaks, из раздела 3.2.13

5.1.1 Автокорреляция

Для проверки наличия автокорреляции используем тест Бройша-Годфри, т.к. этот тест для своего использования требует меньше предположений, чем тест Дарбина-Уотса.

Результат оценки ряда тестом Бройша-Годфри в мат.пакете R:

```
##
##  Breusch-Godfrey test for serial correlation of order up to 1
##
## data:  model1
## LM test = 15.7302, df = 1, p-value = 7.305e-05
```

Как мы видим, p-значение очень низко, нулевая гипотеза об отсутствии автокорреляций отвергается.

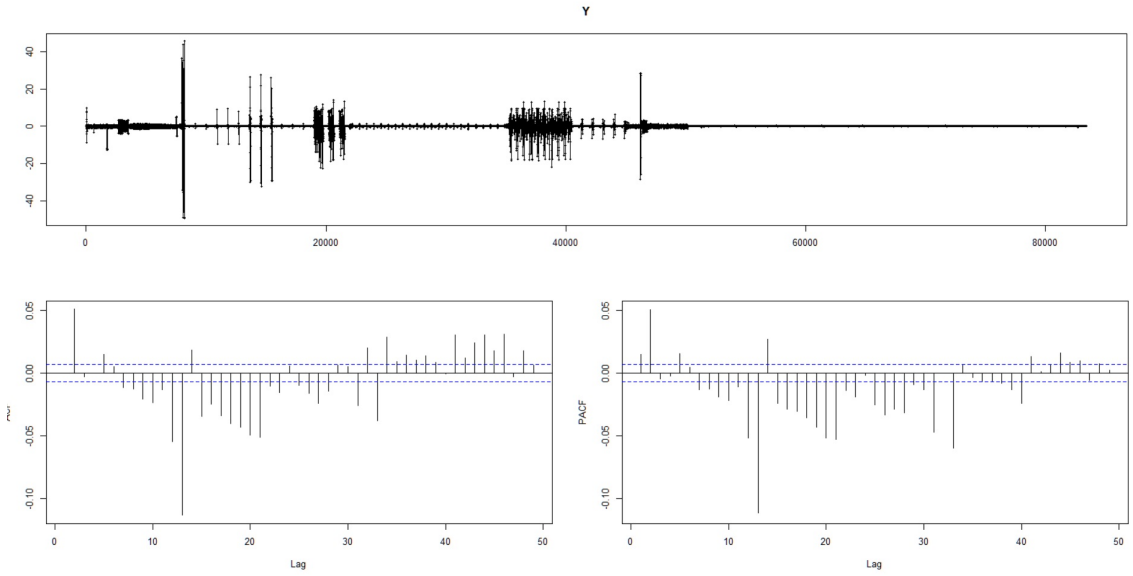

Рис. 5.3: График приращения памяти M(сверху), автокарреляционная функция(слева снизу) и частная автокорреляционная функция(справа снизу).

5.2 Авторегрессионная модель с внешними переменными

5.3 Стационарность

Для проверки временного ряда на стационарность применим тест Дики-Фуллера и KPSS-тест:

```
##
##  Augmented Dickey-Fuller Test
##
## data:  M
## Dickey-Fuller = -57.458, Lag order = 43, p-value = 0.01
## alternative hypothesis: stationary
```

Тест Дики-Фуллера отвергает гипотезу о нестационарности в поль-

зу гипотезы о стационарности.

Применим KPSS-тест:

```
## KPSS Test for Level Stationarity

## data:  M
## KPSS Level = 0.0019, Truncation lag parameter = 66, p-value = 0.1
```

На 5% процентном уровне значимости мы можем сказать, что результат KPSS теста так же говорит о стационарности.

При этом, если показать рисунок 5.3 эксперту, то он вынесет решение о нестационарностя ряда, т.к. там четко просматриваются "кластеры волатильности". Данный факт является основанием для проверки наличия структурных изменений(breaks) в данном ряде (смотри 5.4).

5.4 Breaks

Если взглянуть на рисунок 5.3 возникает вопрос относительно стационарности ряда, не смотря на результаты тестов. Провери наличие breaks из раздела 3.2.13. Т.к. на рисунке 5.3 явно прослеживаются различные "кластеры волатильности"[19]. Проверим данный временной ряд на breaks при помощи QLR теста(смотри 3.2.13).

Результат вычисления F-статистик для каждого τ показан на рисунке 5.4. Черная линия - статистики, красная - критическое значение, равно в нашем случае 3.37[19] при 5% уровне значимости. Таким образом, гипотеза о отсутствии структурных изменений во временом ряде отвергается.

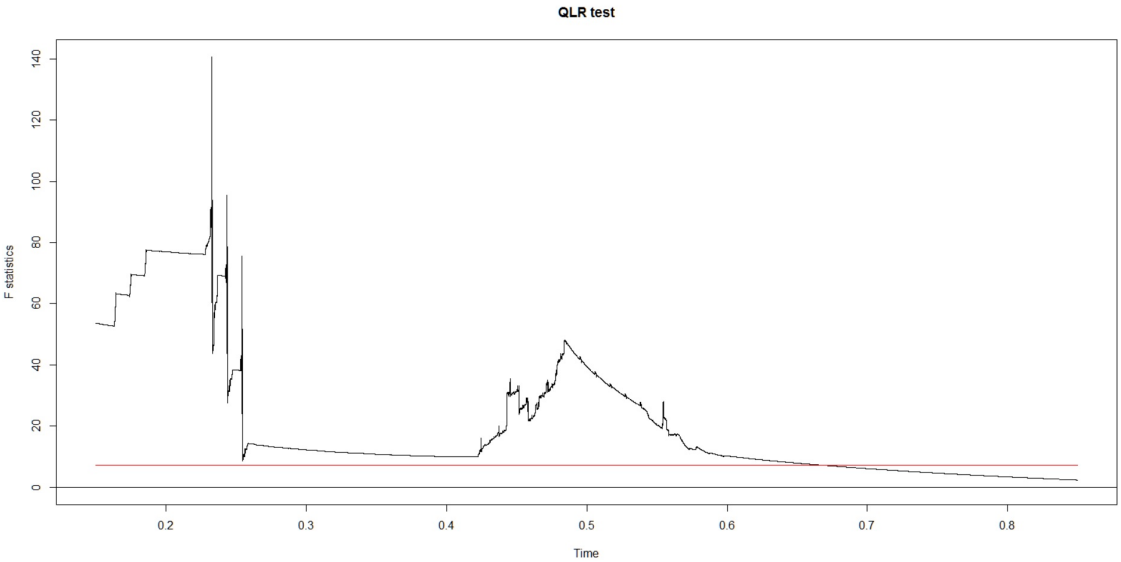

Рис. 5.4: Результат вычисления F-статистик для QLR теста(смотри 3.2.13) на отсутствие структурных изменений временного ряда. Максимум F-статистик и является значением QLR теста.

5.5 MLE регрессии

Так же были уравнение 5.1 было решенно методом максимального правдоподобия:

```
## Maximum likelihood estimation

## Call:
## mle2(minuslogl = LL2, start = list(beta0 = beta[6], beta1 = beta[1],
## beta2 = beta[2], beta3 = beta[3], beta4 = beta[4], beta5 = beta[5],
## mu = mu, sigma = sigma))

## Coefficients:
## Estimate  Std. Error     z value     Pr(z)
## beta0  4.5000e+01  1.7296e-03  26017.3544 < 2.2e-16 ***
## beta1  1.3561e-03  1.8701e-04      7.2515 4.121e-13 ***
```

```
## beta2 -2.4987e-03  1.8734e-03     -1.3338    0.1823
## beta3  3.1371e-03  2.1854e-03      1.4355    0.1512
## beta4  1.4327e-02  2.3457e-03      6.1078 1.010e-09 ***
## beta5  1.2264e-02  2.5218e-03      4.8631 1.155e-06 ***
## mu    -4.5000e+01  1.7296e-03 -26017.3544 < 2.2e-16 ***
## sigma  9.9929e-01  2.4460e-03    408.5334 < 2.2e-16 ***
## ---
## Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

## -2 log L: 236702.5

## AIC: 236718.5
```

Как видно по значению AIC еще хуже чем решенная OLS методом регерссия.

5.6 Выводы

Как было показанно в этой главе временной ряд зависимости памяти от *performance counter* имеет низкой предсказательную способность, низкую R^2_{adj}. Результаты тестов и построения автокорреляционного графика 5.3 говорят о наличии автокорреляций. Тесты на стационарность говорят, что ряд стационарен. Экспертное мнение говорит обратное, что ряд нестационарен. Положительный результат тестирования ряда на breaks. Из этого мы делаем вывод что ряд все-таки не стационарен, просто для установления этого факта недостаточно сведений.

Нестационарность ряда говорить нам о том, что такие техники, как фильтр Кальмана не могут быть применены [20].

В главе 3.2.13 было сказанно, что авторегрессионные модели пло-

хо работают на нестационарных рядах. Регрессия отображает “in average”[19] отношение регрессора и предсказываемой переменной. Фактически, сводя авторегрессию к *Last-state based method* из главы 2.

Т.к. мы не делаем никаких предположений о “сезонности” появления breaks, то метод максимального правдоподобия(смотри 3.2.12) будет тоже не эффективен [19].

Таким образом требуется разработка нового подхода, который будет описан в 6.

ГЛАВА 6

Алгоритмы

6.1 Основная модель управления памятью

Как было показанно в разделе 5 при помощи *performance counters* строится модель с весьма низкой предсказательной способностью. В этом разделе мы рассмотрим простую модель памяти, которая была презентованна в работе [9].

Данная модель представляет собой несколько усовершенствованный *Last state method* из главы 2. Количество памяти, необходимое системы вычисляется по формуле.

$$NeedMemory_t = NeedMemory_{t-1} + \epsilon_t \left(NeedMemory_{t-1}\right) \quad (6.1)$$

Где $NeedMemory_i$ - объем потребляемой памяти в момент времени i. $\epsilon_i \left(NeedMemory_i\right)$ - это функция зависящая от потребления памяти в момент времени i.

В данной работе мы объявляем ϵ_i как:

$$\epsilon_i \left(NeedMemory_i\right) = \alpha NeedMemory_i \quad (6.2)$$

α варьировался от 0.05 до 0.75. Результаты тестирования данной модели будут представленны в главе 7.

Очевидно, что размер извлеченной balloon driver памяти вычисляется по формуле:

$$ballonSize = TotalMemory - NeedMemory \quad (6.3)$$

Алгоритм 1 кратко описывает вычисления размера памяти, которую захватит balloon driver.

```
1 Algorithm: Simple memory controlling
  Input: memory ;                    /* memory consumption now */
  Result: balloonSize
2 useMemory = memory + α × memory;          /* Calculate
  neccessary memory by formulas 6.1, 6.2 */
3 balloonSize = max (totalMemorySize − useMemory, 0);
  /* Calculate by formula 6.3 */
4 return balloonSize
```

Algorithm 1: Simple memory controlling

6.2 Модель управления памятью на основе выделения паттернов потребления

Каждый пользователь, обычно, использует свой компьютер в какой-то свойственной ему манере. Например, на обычном домашнем компьютере человек придя с работы может каждый вечер запускать видео проигрыватель, чтобы посмотреть новую серию любимого сериала. В это время каждый день мы будем наблюдать примерно одинаковый вид потребления памяти. ***Паттерном*** мы будем называть повторяющийся во времени куски временного ряда длинны k.

Идея метода заключается в следующем:

1. После первого системы она начинает накапливать данные относительно интересующего ее ресурса(в нашем случае памяти). Данные представленны в виде отрезков длинны k (семплов), которые отображают последовательное изменение размера рабочего набора виртуальной машины.

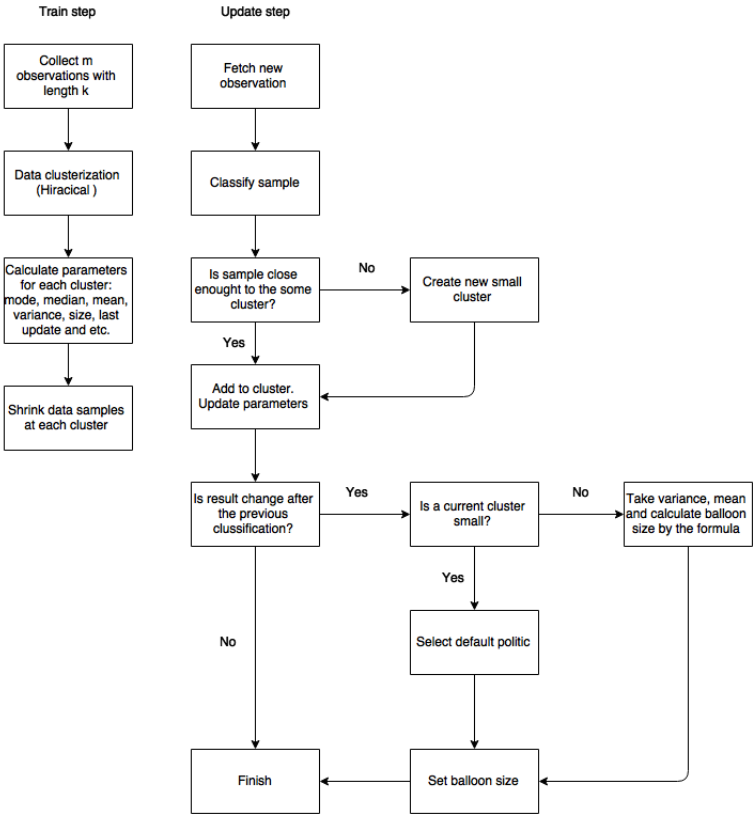

Рис. 6.1: Блок-схема описанного в 6.2 метода. Левая блок-схема демонстрирует пункты $1-4$, правая $5-7$.

2. Когда набранно достаточно семплов - n запускает процесс кластеризации. Алгоритм может быть вырбран среди перечисленных в 3.3. В данной работе для проверки работоспособности идеи был использован алгоритм иерархической кластеризации по принципу buttom-up и k-means. В качестве метрики для кластеризации используется Dynamic Time Warping из 3.3.4. Данная метрика используется потому что она устойчива с сдвигам, в отличии от той же евклидовой метрики.

   Каждый кластер хранит о себе следующие данные:

   - Набор семплов, относящихся к этому классу;
   - Размер класса;
   - Математическое ожидание;
   - Дисперсия;
   - Мода;
   - Медиана;
   - Время последнего изменения;
   - Частота обращения к этому классу.

   Частота обращения к классу вместе с его размером используется для классификации и обновления данных.

3. После кластеризации мы разделяем кластеры на 2 типа:

   - “Большие” - размер больше некоторого N;
   - “Маленькие” - размер меньше некоторого N.

“Маленькие” кластеры, потенциально могут содержать шум. “Большой” - это кластер, размер которого больше или равен $Size_{min}$, “маленькие” все остальные.

4. При необходимости, так же происходит сжатие семплов в каждом кластере для уменьшения занимаемого им пространства.

5. После окончания построения через равные промежутки времени алгоритму подается на вход только что записанный с виртуальной машины новый семпл длины k.

6. Алгоритм производит классификацию и вычисляет наиболее близкий к данному семплу кластер. Если данный семпл находится к кластеру на расстоянии меньшем чем D, то от принадлежит этому кластеру. В таком случае мы добавляем его в соответствующий кластер и обновляем информацию о кластере: пересчитываем время последнего изменения, корректируем значение математического ожидания, моды, медианы и т.д. семплов. Если размер кластера превысил некоторую минимальную величину $Size_{min}$, то он объявляется “большим” кластером. Если семпл находится к ближайшему кластеру на расстоянии большем, чем D - мы создаем новый кластер, который содержит единственный семпл.

7. Мы сравниваем, изменился ли кластер по сравнению с предыдущей классификацией. Если нет, мы ничего не делаем до следующего нового семпла. Иначе, мы смотрим, является ли кластер, к которому был отнесен данный семпл “большим“ или “маленьким”:

- “Большой” - метод возвращает параметры кластера
- “Маленький” - метод возвращает параметры кластера “по-умолчанию“.

Псевдокод шагов $1-4$ представлен в алгоритме 2, шаги $5-7$ в алгоритме 3.

Пусть у нас есть временной ряд потребления памяти как на рисунке 5.1. Тогда, вычислить оценку рабочего набора можно будет по формулам:

$$NeedMemory_t = E\left(Cluster_i\right) + 1.96 \cdot se\left(Cluster_i\right) \tag{6.4}$$

Т.е. мы строим верхнюю границу 5% доверительного интервала для среднего значения памяти в этом кластере. Учитывая неустойчивость среднего значения к выбросам и шумам математическое ожидание можно заменить на моду или медианное значении, т.к. они являются более устойчивыми [19, 39, 40].

Если же у нас ряд изменения памяти по сравнению с предыдущем моментов времени, как на рисунке 5.2, то размер рабочего набора можно оценить по следующей формуле:

$$NeedMemory_t = NeedMemory_{t-1} + E\left(Cluster_i\right) + 1.96 \cdot se\left(Cluster_i\right) \tag{6.5}$$

Опять же вместо мат.ожидания возможно использование моды или медианы.

Тогда размер, занимаемый balloon driver будет вычислен по формуле:

$$BalloonSize = max\left(CommitTotal - NeedMemory_t, 0\right) \tag{6.6}$$

На рисунке 6.1 представленна блок-схема данного метода.

```
1  Algorithm: Pattern Collection
   Input: sampleSet ;        /* set of samples. n samples with
          length k */
   Result: Clusters;    /* Result of sampleSet clusterization
           */
2  Clusters = getClusters(sampleSet, DTW);      /* Clusterize
   sampleSet use a Dynamic Time Warping metrics3.3.4 and
   some clustering algorithm */
3  foreach (c in Clusters) do
       /* Select type for each cluster                    */
4      if (c.getSize () < Size_min) then
5          c.setType (small);
6      else
7          c.setType (big);
8      end
9      c.calculateProperties (); /* Calculate a mean, a mode, a
       median, a variance and etc. */
10     c.compressSamples ()
11 end
```

Algorithm 2: Pattern Collection algorithm

```
1  Algorithm: Pattern Classification
   Input: Clusters ;        /* set of samples. n samples with
          length k */
2  sample
   Result: Cluster
   ;                        /* Cluster that contain the sample */
3  [dist, class] = Classification(sample);
4  if (dist <= D) then
5  |  class.addSample(sample);
6  |  class.updateState ();
7  |  return class;
8  else
9  |  new = createNewClass (sample);
10 |  new.setType(small);
11 |  new.calculateProperties ();
12 |  return new;
13 end
```

Algorithm 3: Pattern Classification algorithm

ГЛАВА 7

Эксперименты

7.1 Тестирование основной модели

Тестирование основной модели проводилось вручную, а так же автоматически при помощи системы PCMark и специального набора тестов компании Parallels.

При мануальном тестировании было заметно замедление работы с файловой системы.

Проведенное автоматизированное тестирование показано в таблице 7.1.

Как видно наиболее большие потери производительности системы Windows 7 и Windows 8 получили при тестировании случайной записи на диск и чтении с диска. Что не удивительно, так как в рамках данной модели система не может отдавать под файловый кеш все свободные ресурсы.

Test	Windows 7, 1CPU, 4Gb	Windows 8, 1CPU, 4Gb
busyloop_test	−2.0%	−2.0%
system_syscall_test	−0.1%	−0.3%
process_exec_test	−2.0%	−2.3%
thread_create_test	−2.0%	−1.8%
io_hdd_seq_rand_rd_test	−99.8%	−99.2%
virtalloc_test	−1.1%	−0.2%
mem_read_test	−1.5%	−5.4%

mem_write_test	−1.6%	−2.9%
mem_pf_read_test	−0.2%	−10.0%
mem_pf_write_test	+0.4%	−9.1%
mem_copy_test	−1.2%	−1.6%

Таблица 7.1: Результаты тестирования простой оценки размера рабочего набора из раздела 6.1.

Для того чтобы ОС Windows не пыталась использовать под файловые кеши всю свободную память, нужно сделать следующее:

1. Поставить в состояние Enable LargeCacheFile в реестре Windows для изменения политики контроля кеша[41];

2. Установить границы размера кеша при помощи системных вызовов[26].

Для тестов размер кеша был установлен в границах от 256Mb до 768Mb.

Результаты тестов с измененной политикой представленны в таблице 7.2. Как можно увидеть из таблицы включение контроля кеша улучшает результат, но все же данная модель не подходит для использования в системах, активно работающих с файлами.

Test	Windows 7, compared to a VM with no memory management Windows 7	Windows 7, compared to the VM with memory management without cache control
busyloop_test	+1.1%	+3.2%
system_syscall_test	−1.0%	−0.9%
process_exec_test	−3.3%	−1.4%
thread_create_test	+3.1%	+5.0%
io_hdd_seq_rand_rd_test	−99.8%	+32.0%
virtalloc_test	−0.4%	+0.7%
mem_read_test	−2.4%	−0.9%
mem_write_test	−2.4%	−0.8%
mem_pf_read_test	−0.3%	−0.0%
mem_pf_write_test	−0.1%	−0.6%
mem_copy_test	−0.9%	+0.3%

Таблица 7.2: Сравнения результатов тестирования простой модели из раздела 6.1 с контролем кеша и без него.

7.2 Тестирование модели на основе выделения паттернов

Прототип метода из раздела 6.2 был реализован в Matlab 2014. Полученные данные о потреблении памяти были разбиты на непересекающиеся наборы обучающих и тестовых. Например, если велись наблюдения в течении 2-х дней, то первый день использовался как источник для обучающих данных, второй день как источник для тестовых. Тестирование построенного алгоритма говорит о том, что алгоритм ошибается в 6.4% случаев.

ГЛАВА 8

Выводы

В данной работе приведено 2 алгоритма оценки размера рабочего набора виртуальной машины.

Дальнейшее направление исследования направленно на встраивание модели из раздела 6.2 в систему Parallels Desktop и проведение тестов.

Одним из возможных направлений исследования является попытка автоматического обнаружения структурных изменений более простыми методами, например, классификация временного ряда на имеющий структурное изменение и не имеющий.

ГЛАВА 9

Заключение

Цель данной магистерской диссертации разработать алгоритм, который оценивает размер рабочего набора виртуальной машины и принимает решение о том, сколько оперативной памяти можно изъять у конкретной виртуальной машины(virtual machine, VM) для перераспределения между другими VM, запущенными на той же хост-машине (host machine).

В работе был проведен анализ потребления виртуальной машиной памяти с статистической точки зрения. Было показано, что регрессии от *performance counters* имеют низкую предсказательную способность. Авторегрессии так же показали плохой результат. Ряды являются нестационарными и с наличием структурных изменений, что говорит о том, что такие методы, как авторегрессии, фильтр Кальмана и т.д. не работоспособны.

Предложенно 2 варианта оценки размера рабочего набора виртуальной машины. Первая модель показала хороший результат, но непригодна для использования в системах, которые активно работают с файловой системой. Прототип второй модели давал ложные предсказания в 6.4% случаев.

Результаты исследований по данной работе опубликованны в [22, 9].

Литература

[1] Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[2] Peter J Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[3] Scott D Lowe. Best practices for oversubscription of cpu, memory and storage in vsphere virtual environments. *VMware's White paper, available at: https://communities. vmware. com/servlet/JiveServlet/previewBody/21181-102-1-28328/vsphereoversubscription-best-practices [1]. pdf*, 2013.

[4] John J Prevost, KranthiManoj Nagothu, Brian Kelley, and Mo Jamshidi. Prediction of cloud data center networks loads using stochastic and neural models. In *System of Systems Engineering (SoSE), 2011 6th International Conference on*, pages 276–281. IEEE, 2011.

[5] Patrick Thibodeau. Data centers are the new polluters@ONLINE, 2015. URL `http://www.computerworld.com/article/2598562/data-center/data-centers-are-the-new-polluters.html`.

[6] Truong Vinh Truong Duy, Yukinori Sato, and Yasushi Inoguchi. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[7] Anna Melekhova. Machine learning in virtualization: Estimate a virtual machine's working set size. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 863–870. IEEE, 2013.

[8] Carl A Waldspurger. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review*, 36(SI):181–194, 2002.

[9] Melekhova A. and Markeeva L. Estimating working set size by guest os performance counters means. *CLOUD COMPUTING 2015*, page 48, 2015.

[10] Sheng Di, Derrick Kondo, and Walfredo Cirne. Host load prediction in a google compute cloud with a bayesian model. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 21. IEEE Computer Society Press, 2012.

[11] Anderson Ravanello, Luis Villalpando, Jean-Marc Desharnais, Alain April, and Abdelouahed Gherbi. Associating performance measures with perceived end user performance. *CLOUD COMPUTING 2015*, page 125, 2015.

[12] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE, 2010.

[13] Kinshuk Govil, Dan Teodosiu, Yongqiang Huang, and Mendel

Rosenblum. Cellular disco: Resource management using virtual clusters on shared-memory multiprocessors. *ACM SIGOPS Operating Systems Review*, 33(5):154–169, 1999.

[14] Andrea C Arpaci-Dusseau and Remzi H Arpaci-Dusseau. Information and control in gray-box systems. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 43–56. ACM, 2001.

[15] Pei Cao, Edward W Felten, and Kai Li. Implementation and performance of application-controlled file caching. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 13. USENIX Association, 1994.

[16] Carl A Waldspurger and William E Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 1. USENIX Association, 1994.

[17] Carl Waldspurger, William E Weihl, et al. An object-oriented framework for modular resource management. In *Object-Orientation in Operating Systems, 1996., Proceedings of the Fifth International Workshop on*, pages 138–143. IEEE, 1996.

[18] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

[19] James H Stock and Mark W Watson. *Introduction to econometrics*, volume 104. Addison Wesley Boston, 2003.

[20] William Bell and Steven Hillmer. Initializing the kalman filter for nonstationary time series models. *Journal of Time Series Analysis*, 12(4):283–300, 1991.

[21] Stefan Frey, Simon Disch, Christoph Reich, Martin Knahl, and Nathan Clarke. Cloud storage prediction with neural networks. *CLOUD COMPUTING 2015*, page 67, 2015.

[22] Мелехова А.Л. Маркеева Л.Б. Тормасов А.Г. Однородность виртуализационных событий, порожденных различными операционными системами. *Труды МФТИ*, (1), 2014.

[23] Sayanta Mallick, Gaétan Hains, and Cheikh Sadibou Deme. A resource prediction model for virtualization servers. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 667–671. IEEE, 2012.

[24] US Air Force. Analysis of the intel pentium's ability to support a secure virtual machine monitor. 2000.

[25] Part Guide. Intel® 64 and ia-32 architectures software developer's manual. 2010.

[26] Mark E Russinovich, David A Solomon, and Jim Allchin. *Microsoft Windows Internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000*, volume 4. Microsoft Press Redmond, 2005.

[27] James Bernard Ramsey. Tests for specification errors in classical linear least-squares regression analysis. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 350–371, 1969.

[28] Никита Вячеславович Артамонов. Введение в эконометрику. *НВ Артамонов.-: МЦНМО*, 2011.

[29] K Person. On lines and planes of closest fit to system of points in space. philiosophical magazine, 2, 559-572, 1901.

[30] Halbert White. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica: Journal of the Econometric Society*, pages 817–838, 1980.

[31] Stephen M Goldfeld and Richard E Quandt. Some tests for homoscedasticity. *Journal of the American statistical Association*, 60(310):539–547, 1965.

[32] Лбов Г.С. Бериков В.Б. Современные тенденции в кластерном анализе. *Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы*, 2008.

[33] Andrea Vattani. K-means requires exponentially many iterations even in the plane. *Discrete & Computational Geometry*, 45(4):596–616, 2011.

[34] Ethem Alpaydin. *Introduction to machine learning.* MIT press, 2014.

[35] Oded Maimon and Lior Rokach. *Data mining and knowledge discovery handbook*, volume 2. Springer, 2005.

[36] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything

you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer, 2004.

[37] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49, 1978.

[38] Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 23(1):67–72, 1975.

[39] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[40] Anthony McCluskey and Abdul Ghaaliq Lalkhen. Statistics ii: Central tendency and spread of data. *Continuing Education in Anaesthesia, Critical Care & Pain*, 7(4):127–130, 2007.

[41] O. Pentakalos M. Friedman. *Windows 2000 Performance Guide*. 2002.