DOI: 10.15514/ISPRAS-2019-31(4)-2

Автоматизация обнаружения и анализа ошибок в гиперконвергентных системах

Д.В. Силаков, ORCID: 0000-0001-9175-6943 <dsilakov@virtuozzo.com>
ООО «Виртуоззо Рисерч»,
127273, Россия, г. Москва, ул. Отрадная, д. 2Б, стр. 9

Аннотация. Статья посвящена проблеме выявления и оперативного анализ ошибок, возникающих при эксплуатации гиперконвергентных систем. Одним из подходов к организации гиперконвергентных систем является установка на каждый физический сервер отдельного экземпляра операционной системы (ОС), несущей в себе средства виртуализации и инструментарий для администрирования и использования распределенного хранилища данных. Возникновение ошибок возможно как на уровне отдельного экземпляра ОС, так и на уровне всего кластера, Например, некорректные команды управляющих элементов с одного узла инфраструктуры могут вызвать сбой ПО на другом узле. Кроме того, ошибки со стороны подсистем кластера могут спровоцировать нештатные ситуации внутри виртуальных машин. Сложность архитектуры гиперконвергентных систем обуславливает сложность анализа возникающих в них ошибок. Для упрощения такого анализа и повышения его эффективности необходима автоматизация процесса обнаружения проблем и сбора данных, необходимых для их изучения и исправления. Рассматриваются подходы к автоматизации подобных процессов в существующих ОС и предлагаются способы их адаптации к системам, использующим распределенное хранилище данных и виртуализацию. Описывается опыт применения адаптированных решений в продуктах Virtuozzo.

Ключевые слова: обнаружение ошибок; виртуализация; хранилище данных

Для цитирования: Силаков Д.В. Автоматизация обнаружения и анализа ошибок в гиперконвергентных системах. Труды ИСП РАН, том 31, вып. 4, 2019 г., стр. 29-38. DOI: 10.15514/ISPRAS-2019-31(4)-2

Automated Error Detection and Analysis in Hyperconverged Systems

D.V.Silakov, ORCID: 0000-0001-9175-6943 <dsilakov@virtuozzo.com>
Virtuozzo,
Seattle, USA (HQ) 110 110th Ave NE #410, Bellevue, WA 98004

Abstract. The paper is devoted to the problem of early error detection and analysis in hyperconverged systems. One approach to organizing hyperconverged systems is to install on each physical server a separate instance of an operating system (OS) that carries virtualization tools and tools for administering and using a distributed data warehouse. Errors can occur both at the level of a single OS instance and at the level of the entire cluster. For example, incorrect control element commands from one infrastructure node can cause software failure on another node. In addition, errors from the subsystems of the cluster can provoke abnormal situations inside virtual machines. The complexity of the architecture of hyperconverged systems makes it difficult to analyze the errors that occur in them. To simplify such an analysis and increase its effectiveness, it is necessary to automate the process of detecting problems and collecting data necessary for their study and correction. Existing approaches for automation of error detection are described and various improvements are suggested to adopt them for systems where distributed storage and virtualization technologies are actively used. Improvements include log collection from the whole cluster just after the error occurred, additional analysis of guest operating system behaviour inside virtual machines, usage of a knowledge base for automated crash recovery and duplicate detection. Finally, a real-life scenario of error handling process in Virtuozzo company products is described starting from error detection and ending with fix deployment.

Keywords: error detection; virtualization; data storage

For citation: Silakov D.V. Automated Error Detection and Analysis in Hyperconverged Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 4, 2019. pp. 29-38 (in Russian). DOI: 10.15514/ISPRAS-2019-31(4)-2

1. Введение

Выявление проблем в ходе эксплуатации ПО и оперативное реагирование на их возникновение – задача, которая вряд ли когда-нибудь потеряет актуальность. Мониторинг и анализ нештатных ситуаций необходимы как при реальной эксплуатации, так и при различного вида тестировании на всех стадиях разработки.

К настоящему времени создано немало средств автоматизации подобного мониторинга, однако новые тенденции в миру ПО предъявляют новые требования к инструментарию. К одной из таких тенденций относится создание гиперконвергентных инфраструктур, которые подразумевают использование программных средств для объединения ресурсов множества серверов в кластер, являющийся с точки зрения системных администраторов и пользователей единой средой с централизованным управлением.

Ресурсы серверов включают прежде всего вычислительные мощности и дисковое пространство. В гиперконвергентной системе устройства хранения данных множество машин объединяется в сетевое хранилище (возможно, с дублированием данных в целях повышения отказоустойчивости), а вычислительные ресурсы используются для создания и запуска виртуальных машин. Именно последние занимаются выполнением задач конечных пользователей. Такой подход позволяет гибко и в то же время эффективно распределять физические ресурсы между различными прикладными задачами.

Одним из подходов к организации гиперконвергентных систем является установка на каждый физический сервер отдельного экземпляра операционной системы (ОС), несущей в себе средства виртуализации и инструментарий для администрирования и использования распределенного хранилища данных.

Возникновение ошибок возможно как на уровне отдельного экземпляра ОС, так и на уровне всего кластера – например, некорректные команды управляющих элементов с одного узла инфраструктуры могут вызвать сбой ПО на другом узле. Кроме того, ошибки со стороны подсистем кластера могут спровоцировать нештатные ситуации внутри виртуальных машин.

Сложность архитектуры гиперконвергентных систем обуславливает сложность анализа возникающих в них ошибок. Для упрощения такого анализа и повышения его эффективности необходима автоматизация процесса обнаружения проблем и сбора данных, необходимых для их изучения и исправления.

2. Существующие подходы к автоматизации обработки нештатных ситуаций

Существующие подходы к обнаружению нештатных ситуаций можно разделить на две основных категории:

- выявление ошибок непосредственно в момент их возникновения;
- выявления факта возникновения проблемы на основе анализа системных журналов.

2.1 Оперативное выявление проблем

Применение первого способа позволяет произвести анализ системы и сбор потенциально полезной для анализа проблемы информации «по горячим следам». Например, можно

сделать снимок памяти проблемного процесса, сохранить контекст его выполнения и так далее.

Однако реализация такого подхода сильно зависит от того, какого рода ошибки необходимо выявлять. Каждый тип нештатной ситуации может потребовать разработки отдельного инструментального средства для его оперативного обнаружения.

Например, широко применяемые в Linux инструментальные наборы ABRT [1] и Apport [2] позволяют отслеживать следующие виды нештатных ситуаций:

- ошибки сегментации («Segmentation Fault») в бинарных программах – как правило, вызванные некорректной работой с памятью;
- возникновение исключений в программах на Python, Ruby и Java;
- аварийное завершение работы ядра Linux с последующей перезагрузкой системы, а также сообщения «OOPS» от ядра, сигнализирующие о нештатной ситуации, выявленной самим ядром (которая могла и не повлечь остановки машины);
- возникновение проблем при работе графической подсистемы (X-сервера).

Для обнаружения каждой из указанных проблем реализована отдельная утилита, использующая специфические средства ОС и системного ПО. В частности:

- ошибки работы с памятью регистрируются с помощью ядра Linux, посредством добавления обработчика в файл /proc/sys/kernel/core_pattern [3];
- для обнаружения исключений в Java-программах в среде исполнения Java (Java Runtime Environment) регистрируется соответствующий обработчик, использующий Java Native Interface;
- схожий механизм используется для обнаружения проблем в скриптах на Python и Ruby – их среды исполнения также позволяют регистрировать сторонние обработчики исключений;
- аварийное завершения работы ядра Linux, приведшие к перезагрузке машины, определяется по наличию файла vmcore (образа памяти ядра на момент завершения работы, автоматически создаваемого при возникновении нештатной ситуации);
- прочие нештатные ситуации, связанные с ядром Linux (к таковым также относятся потенциальные проблемы с оборудованием, о которых сообщает ядро), отслеживаются путем мониторинга файла /proc/kmsg, предоставляющего прямой доступ к сообщениям ядра;
- для обнаружения проблем графической подсистемы ведется постоянный мониторинг файла журнала /var/log/Xorg.0.log и анализ добавляемой в него информации.

К достоинствам данного подходя относится возможность собрать информацию, которая доступна только в течение короткого промежутка времени после возникновения инцидента, но которая будет крайне полезна для его анализа. К подобным данным относится снимок памяти упавшего процесса, перечень открытых им файлов и сетевых соединений и прочая информация, которая находится только в оперативной памяти машины.

Архитектура инструментальных средств наподобие ABRT и Apport является модульной и позволяет добавлять обработчики произвольных событий – необходимо только разработать соответствующее инструментальное средство. Необходимость такой разработки и является основным фактором, ограничивающим применение подхода.

Для многих приложений единственный способ определить нештатную ситуацию в их работе (не вызывающую аварийного останова программы, но критическую для ее функционирования) заключается в анализе журналов – как общесистемных, так и их собственных. Разрабатывать и запускать инструменты такого анализа для каждого приложения в системе может оказаться слишком накладно (особенно если стоит цель

мониторинга журналов в реальном времени) – и тогда может быть использован альтернативный подход.

2.2 Выявление ошибок на основе анализа журналов

Другое популярное решение основано на том, что в большинстве программных систем ведутся журналы различных событий, в число которых обычно входят и любые нештатные ситуации. Анализируя журналы событий, можно определить и факт возникновения тех или иных инцидентов, требующих внимания разработчиков и системных администраторов. Можно осуществлять как мониторинг журналов в реальном времени, реагируя на каждую новую запись, так и периодический анализ данных, появившихся за определенный промежуток времени.

В настоящее время существует множество программных средств, занимающихся анализом и фильтрацией журналов с целью выявления заданных событий. При этом многие решения (например, Graylog [4]) допускают обработку журналов со множества машин и имеют расширяемую архитектуру, позволяющую оперативно добавлять формальные описания интересующих пользователя системы событий.

Наряду с однозначными формальными правилами фильтрации событий (например, с использованием регулярных выражений для выделения сообщений об ошибках), для выявления нештатных ситуаций могут применяться нечеткие правила и правила, выведенные на основе машинного обучения [5].

Помимо гибкости в добавлении описаний и масштабируемости, к достоинствам таких систем стоит отнести возможность повторного анализа уже имеющихся журналов при добавлении или обновлении описания события для анализа. Основным же недостатком в большинстве случаев является отсутствие в журналах информации, необходимой для детального изучения проблемы.

Например, аварийное завершение приложений в дистрибутивах Linux зачастую может остаться вне поля зрения системных журналов. Но даже если факт аварийной остановки будет отмечен, то в журналы редко помещается детальная информация – такая, как снимок памяти процесса. Впрочем, для приложений, использующих для работы специфическую среду исполнения (в частности – программ, написанных на языках Go, Java, Python и ряде других), в системные журналы может помещаться трасса вызова функций, приведших к возникновению нештатной ситуации.

Тем не менее, в общем случае подход с анализом журналов в плане полноты информации сильно уступает целенаправленному сбору данных об инциденте в момент его возникновения.

Ниже мы рассмотрим возможные усовершенствования существующих подходов к обработке инцидентов «по горячим следам», которые позволят повысить эффективность анализа проблем.

3. Обход кластера для сбора информации

Большинство автоматических средств сбора информации о нештатных ситуациях в момент их возникновения ограничиваются рамками ОС, в которой они работают. В гиперконвергентных системах такой подход может оказаться недостаточен. Такие системы являются распределенными, и ОС на отдельно взятой машине не является изолированной и самодостаточной – в ней могут выполняться служебные процессы, контролируемые управляющими элементами с других узлов кластера.

Для воспроизведения полной картины инцидента, связанного с подобными служебными процессами, может потребоваться информация и от управляющих узлов. Например, в системах, реализующих отказоустойчивость сервисов посредством их дублирования,

некорректное поведение служебной программы может быть вызвано поступлением противоречивых команд от разных управляющих процессов.

Обход узлов кластера и сбор данных с удаленных машин – более дорогая операция, чем анализ в рамках одного физического сервера. Поэтому перед обращением на удаленные машины необходимо определить, есть ли в этом смысл.

Одним из возможных способов выяснить целесообразность обхода является составление списка процессов, ошибки в которых могут быть вызваны удаленными узлами кластера. В этот список могут входить служебные сервисы, обслуживающие инфраструктуру кластера, а также любые программы, обращающиеся напрямую к удаленным узлам системы. Если инцидент произошел с процессом, не входящим в этот список, то собирать данные с удаленных узлов не надо.

Еще большей точности можно достигнуть, проанализировав, с какими узлами и сервисами кластера взаимодействовал проблемный процесс непосредственно перед возникновением аварийной ситуации.

4. Анализ ошибок виртуализации

Помимо распределенной природы многих сервисов, важным аспектом гиперконвергентных систем является использование виртуальных машин (ВМ). В идеале каждая ВМ с точки зрения пользователя равнозначна отдельной физической машине. Однако изоляция процессов одной ВМ от другой реализуется с использованием программных средств (гипервизора либо ядра ОС при использовании контейнерной виртуализации), в которых могут содержаться ошибки, приводящие либо к нарушению изоляции, либо дезориентации гостевой операционной системы.

Проблемы в подсистеме виртуализации могут фатально сказаться как на виртуальных окружениях, так и на хост-системе, в которой они запущены. Ввиду возможности таких ошибок отдельной обработки заслуживают следующие ситуации:

- аварийный останов виртуальной машины;
- ошибка в сервисе, входящем в систему виртуализации на хост-системе.

В обоих случаях необходимо собрать данные как о событиях, происходивших на сервере в приложениях виртуализации, так и о том, что происходило в этот момент внутри ВМ. К потенциально полезным данным изнутри ВМ относятся:

- сведения о нештатных ситуациях внутри гостевой ОС (в частности, аварийное завершение работы приложений и ядра ОС);
- список процессов, которые были активны на момент инцидента;
- список установленного ПО.

Список процессов и приложений может помочь в определении потенциально вредоносного ПО, работавшего внутри ВМ, которое и могло стать причиной инцидента [6].

Информация изнутри ВМ может быть извлечена либо посредством гостевого агента (при условии, что ВМ запущена) либо непосредственно с образа ее жесткого диска (при условии, что в гостевой системе не используется шифрование дискового пространства). В случае, если в результате инцидента произошло аварийное выключение ВМ, система виртуализации может предоставить снимок памяти всех процессов гостевой ОС для дальнейшего анализа.

5. Выявление одинаковых ошибок

Перед сдачей программного обеспечения в эксплуатацию большинство разработчиков проводят его тщательное тестирование. Тем не менее, определенный процент ошибок в программах остается, но для их проявления нужны некоторые специфические условия - определенное сочетание переменных среды и аргументов, нехватка оперативной памяти или

дискового пространства и тому подобное. Как следствие, подобные ошибки встречаются относительно редко, но при массовом использовании продукта общее число инцидентов может стать значительным.

В частности, в гиперконвергентных системах многие машины зачастую имеют идентичную конфигурацию и занимаются схожими задачами. Велика вероятность того, что если ошибка возникла на одном из серверов, то она может воспроизвестись и на других серверах. Знание о том, что проблеме подвержено множество машин, полезно, однако проводить тщательный анализ каждой из них излишне.

Проверка того, являются ли две ошибки дубликатами или нет, – отдельная задача, для которой существует множество возможных решений. Одно и наиболее простых – это прямое сравнение трасс вызова функций. Этот метод может быть не очень эффективен в случае сложных приложений (в частности – многопоточных), но существует немало решений, производящих нечеткое сравнение трасс, в том числе использующих машинное обучение для поиска дубликатов – см., например, работы [7] и [8].

Большинство методов определения дубликатов применимо независимо от того, принадлежат ли трассы ошибкам, случившимся на одной физической машине или на разных машинах. Однако в настоящее время они применяются на приемнике, куда поступают отчеты об ошибках. Если же мы хотим избежать сбора излишних данных на стороне пользовательской системы (и не посылать излишнего количества отчетов), то выявление дубликатов должно производиться на стороне пользователя.

В ряд существующих инструментов (в частности, ABRT) уже встроен механизм отсеивания дубликатов – для этого в системе хранится история уже обработанных ранее ошибок, и при обнаружении новой проблемы она первым делом сравнивается с теми, что есть в истории.

В случае наличия кластера взаимосвязанных машин логичным усовершенствованием данного подхода будет создание единого (в рамках кластера) хранилища с историей ошибок. При возникновении нового инцидента на одном из узлов кластера инструментарий анализа проблем сможет сверяться с этим хранилищем и принимать решение – производить детальный анализ и сбор данных о проблеме либо проигнорировать ее как уже обработанную.

Помимо истории ошибок конкретного кластера, для обнаружения дубликатов может быть задействована и централизованная база знаний об ошибках на стороне разработчиков или системных администраторов, о которой пойдет речь в следующем разделе.

6. Обратная связь

Большинство программ, отслеживающих возникновение нештатных ситуаций, играют исключительно пассивную роль – они только отправляют извещение о проблеме и никак не вмешиваются в работу системы. Однако с точки зрения пользователя важно не только обнаружить проблему и изучить ее причины, но и оперативно ее устранить.

Современные программные комплексы предоставляют набор примитивов, позволяющих в ряде случаев автоматически восстанавливать функционал системы. Например, при аварийном останове некоторого сервиса он может автоматически перезапускаться средствами ОС, при нехватке места на диске могут удаляться наименее ценные файлы и так далее.

Такие средства, предоставляемые ОС, претендуют на универсальность и, как следствие, во многих случаях не могут помочь, поскольку не учитывают специфику конкретной проблемы. Так, если сервис аварийно останавливается из-за поврежденного файла конфигурации, то стартовать сервис до исправления этого файла бессмысленно.

Какие именно действия необходимо произвести для возвращения системы к штатному режиму работы, зависит от конкретной программы и от того, что с ней произошло. Заложить все возможные сочетания проблем и их решений непосредственно в систему при ее развертывании не представляется возможным – ведь множество таких сочетаний растет в

ходе жизненного цикла ПО как за счет выявления новых потенциальных проблем, так и за счет разработки новых путей для их решений.

Перспективным решением этой проблемы является использование базы знаний, содержащей информацию о нештатных ситуациях, которые могут возникнуть в ходе эксплуатации системы, а также о методах их решения.

Подобные базы уже поддерживаются многими проектами, однако описания проблем и их решений в них содержится в человекочитаемой, зачастую – неформальной форме. Для использования этих баз в программных средствах обнаружения ошибок необходимо выделить формальные признаки, которые характеризуют каждую проблему и наличие которых можно определить автоматически.

К подобным признакам можно отнести:

- имя процесса, вызвавшего нештатную ситуацию;
- имя и версию приложения, частью которого является процесс;
- трассу вызова функций, приведших к аварийной ситуации;
- значение переменных среды, которые могли повлиять на ход работы приложения.

Какие именно признаки учитывать – зависит от каждого конкретного случая. Например, в случае ошибок ядра имя приложения всегда одинаково, а вот версия может иметь значение.

Отметим, что все перечисленные признаки являются строками либо наборами строк. В качестве уникального идентификатора проблемы в базе знаний можно использовать хэш-сумму всех признаков, характеризующих проблему.

Решение проблемы также должно быть представлено в виде, пригодном для автоматического применения. Наиболее простым подходом к этому является оформление решения в виде исполнимого файла, который достаточно запустить на целевой системе.

При наличии формализованной базы знаний автоматизированные средства обнаружения нештатных ситуаций могут использовать ее для поиска и применения решения возникшего инцидента. Доступ к базе может осуществляться как удаленно через Интернет, так и посредством использования локальной и регулярно обновляемой копии.

В любом случае особое внимание должно быть уделено проверке подлинности базы – ведь решение проблемы подразумевает выполнение ряда действий на целевой системе и компрометация базы знаний может быть использована злоумышленниками с целью выполнения вредоносного кода.

7. Жизненный цикл ошибки в системах Virtuozzo

Флагманские продукты компании Virtuozzo – Virtuozzo 7 и Virtuozzo Infrastructure Platform – являются типичными представителями гиперконвергентных систем, предоставляющих конечным пользователям виртуальные машины и распределенное отказоустойчивое хранилище для размещения данных этих ВМ.

Для обнаружения и обработки нештатных ситуаций в указанных продуктах используется связка программ ABRT и libreport, которая применяется во многих дистрибутивах Linux, использующих для управления ПО инструментарий RPM. ABRT предоставляет набор средств для обнаружения определенных видов проблем. libreport дополняет их средствами составления отчетов об обнаруженных проблемах для их последующего анализа, а также доставки этих отчетов.

При выявлении нештатной ситуации ABRT в совокупности с libreport собирают сведения, потенциально полезные для анализа (имя программы, ее опции, снимок памяти и прочее), и оповещают о них заинтересованных лиц – например, системных администраторов или разработчиков.

Связка ABRT и libreport хорошо зарекомендовала себя во многих дистрибутивах Linux, однако потребовала доработки для эффективного использования в решениях Virtuozzo. В

частности, уже добавлены либо планируются к реализации следующие функции, описанные в данной статье:

- сбор необходимых данных не только с машины, где обнаружена проблема, но и со всего кластера;
- обнаружения проблем в гостевых ОС виртуальных машин, которые могут быть вызваны ошибками виртуализации;
- анализ ошибок подсистемы виртуализации, которые могли быть вызваны гостевой ОС;
- динамический выбор информации, помещаемой в журнал об ошибке, в зависимости от деталей инцидента (например, включение журналов тех или иных сервисов в зависимости от того, в каком компоненте возник инцидент);
- использование единого для кластера перечня уже случавшихся ошибок с целью определения дубликатов.

Кроме того, рассматривается возможность реализации «обратной связи», когда приемник отчетов не просто получает сведения об ошибке, но и возвращает список действий, которые можно в автоматическом режиме выполнить на проблемной машине для устранения последствий (например, удалить либо воссоздать необходимые файлы, перезапустить сервис и так далее).

Для реализации функционала обратной связи планируется провести предварительное исследование инцидентов, обнаруженных за время работы инструментов автоматического сбора ошибок. В рамках исследования необходимо выяснить, для каких случаев была необходима процедура восстановления системы после ошибки и в каком проценте ситуаций эта процедура могла бы быть проведена автоматически. Эти данные позволят оценить целесообразность дальнейшей разработки.

Сочетание ABRT и libreport является лишь одним из звеньев цепочки анализа ошибок в Virtuozzo. Полностью процесс обработки проблемы, возникшей в работающей системе, выглядит следующим образом.

1) ABRT обнаруживает возникновение проблемы.
2) libreport совместно с ABRT собирают информацию для отладки. На это стадии также работают дополнительные утилиты Virtuozzo, собирающие сведения, специфичные для продуктов компании.
3) Для обнаруженной проблемы строится идентификатор, который является одним и тем же для одинаковых проблем, возникших на разных машинах или на одной и той же машине в разное время. Например, в случае аварийного останова бинарной программы таким идентификатором может быть хэш-сумма от конкатенации имени упавшего процесса и стека вызовов функций, приведших к падению.
4) Идентификатор проблемы отправляется на сервер-приемник отчетов, который смотрит, приходили ли уже отчеты о проблемах с таким идентификатором или нет.
5) Если подобные отчеты уже были, то приемник отчетов на своей стороне отмечает появление еще одной машины, на которой возникла проблема.
6) Если отчет новый, то на приемник дополнительно отсылается вся информация, собранная на шаге 2. Идентификатор проблемы запоминается в базе данных об ошибках, а информация о проблеме передается на следующий шаг – автоматическому анализатору отчетов.
7) Анализатор строит уточненный отчет об ошибке с использованием данных, не доступных на пользовательских машинах. Например, для бинарных программ на основе отладочной информации и исходных кодов строятся трассы вызовов функций с указанием значения параметров и с привязкой к коду. Построить такой отчет на стороне пользователя невозможно ввиду отсутствия доступа к исходному коду приложений.

8) На основе уточненного отчета автоматически оформляется сообщение об инциденте в системе контроля ошибок.

Таким образом, разработчики получают сообщения об ошибках в привычной им среде и с данными, достаточными для первичного анализа проблемы. Для более глубокого изучения разработчикам предоставляется сервис, создающий контейнер с окружением, максимально приближенным к тому, где возникла проблема (в частности, с точно совпадающим набором установленных приложений и их версий при условии, что в системе не устанавливались сторонние компоненты). Такой контейнер может быть использован для воспроизведения ошибки и для отладки программы.

Указанная схема используется как для мониторинга ошибок на реальных системах клиентов компании, так и на тестовых установках при проведении различного рода проверок. За 1 год использования была накоплена следующая статистика:

- среднее количество отчетов в день: 25000 (99.9% из них поступает с тестовых стендов);
- средний размер отчета (в сжатом виде): 100 Мб;
- общий размер накопленных отчетов (без учета сообщений, признанных дубликатами и не сохраненных в базе): 2,5 Тб
- сообщений об ошибках, автоматически созданных на основе отчетов в системе учета ошибок: 3000 (что составляет ~10% от общего количества инцидентов, заведенных в системе за тот же период);
- исправлено ошибок в продуктах: 400 (~5% от общего количества исправлений, сделанных на основе сообщений об ошибках).

На основе собранной статистики был сделан вывод о целесообразности дальнейшего использования и развития инструментария. Одним из основных направлений развития представляется более скрупулезный выбор информации, добавляемый в отчет об ошибке. В частности, предлагается для каждого ключевого приложения составить свой перечень данных, которые необходимо собрать при возникновении в нем ошибки. Также возможны дальнейшие улучшения определения дубликатов – например, за счет использования машинного обучения.

8. Заключение

С ростом сложности программного обеспечения растет и сложность анализа возникающих в нем ошибок. Упростить анализ можно путем его автоматизации – в частности, посредством автоматического сбора данных, необходимых для изучения проблемы.

Для получения наиболее полной информации сбор данных необходимо проводить непосредственно после возникновения инцидента. В настоящее время уже существуют хорошо зарекомендовавшие себя подходы и инструментальные средства для решения этой задачи, но большинство из них нацелено на работу в рамках одной операционной системы на одном компьютере.

В то же время современные тенденции состоят в развитии гиперконвергентных систем, объединяющих множество физических машин в единое целое и предоставляющей конечным пользователям виртуализированные ресурсы. В рамках таких систем ошибка на одной машине может быть вызвана некорректными действиями других частей системы. Поэтому для эффективного использования в гиперконвергентных средах имеющиеся решения должны быть доработаны с учетом особенностей этих сред.

В данной статье мы предложили ряд таких усовершенствований, большинство из которых уже реализовано нами в продуктах Virtuozzo 7 и Virtuozzo Infrastructure Platform. За более чем годовой период использования реализованные инструментальные средства доказали свою эффективность, позволив выявить и исправить заметное количество ошибок, не перегружая при этом разработчиков ложными срабатываниями и дублирующимися сообщениями.

Список литературы / References

[1]. Doleželová M., Muehlfeld M. et al. Automatic Bug Reporting Tool (ABRT). Deployment, Configuration, and Administration of Red Hat Enterprise Linux 7. Chapter 25 (online). Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/ch-abrt.

[2]. Apport. Ubuntu Wiki (online). Доступно по ссылке: https://wiki.ubuntu.com/Apport

[3]. How to set process core file names. Red Hat Customer Portal (online). Available at: https://access.redhat.com/solutions/901293

[4]. Силаков Д.В. Открытое решение Graylog. Сбор и анализ событий в сетях промышленных масштабов. Системный администратор, № 3, 2019 г., стр. 24-29 / Silakov D.V. Open Graylog Solution. Collection and analysis of events in networks of industrial scale. System Administrator, № 3, 2019, pp. 24-29 (in Russian).

[5]. Du Min, Li Feifei, Zheng Guineng, and Srikumar Vivek. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1285-1298.

[6]. P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, vol. 28, issues 1-2, 2009, pp. 18–28.

[7]. K.K. Sabor, A. Hamou-Lhadj, and A. Larsson. DURFEX: A Feature Extraction Technique for Efficient Detection of Duplicate Bug Reports. In Proc. of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 240-250.

[8]. R.P. Gopalan and A. Krishna. Duplicate Bug Report Detection Using Clustering. In Proc. of the 2014 23rd Australian Software Engineering Conference, 2014, pp. 104-109.

Информация об авторе / Information about the author

Денис Владимирович СИЛАКОВ – кандидат физико-математических наук, старший системный архитектор компании Virtuozzo, отвечает за автоматизацию процессов разработки и сопровождения продуктов компании. Сфера научных интересов: автоматизация тестирования, управление требованиями, автоматизация разработки ПО.

Denis Vladimirovich SILAKOV – Ph.D. in Physical and Mathematical Sciences, Senior system architect in the Virtuozzo company, responsible for automation of development and maintanence of company products. Research interests: test automation, requirements, software development automation.