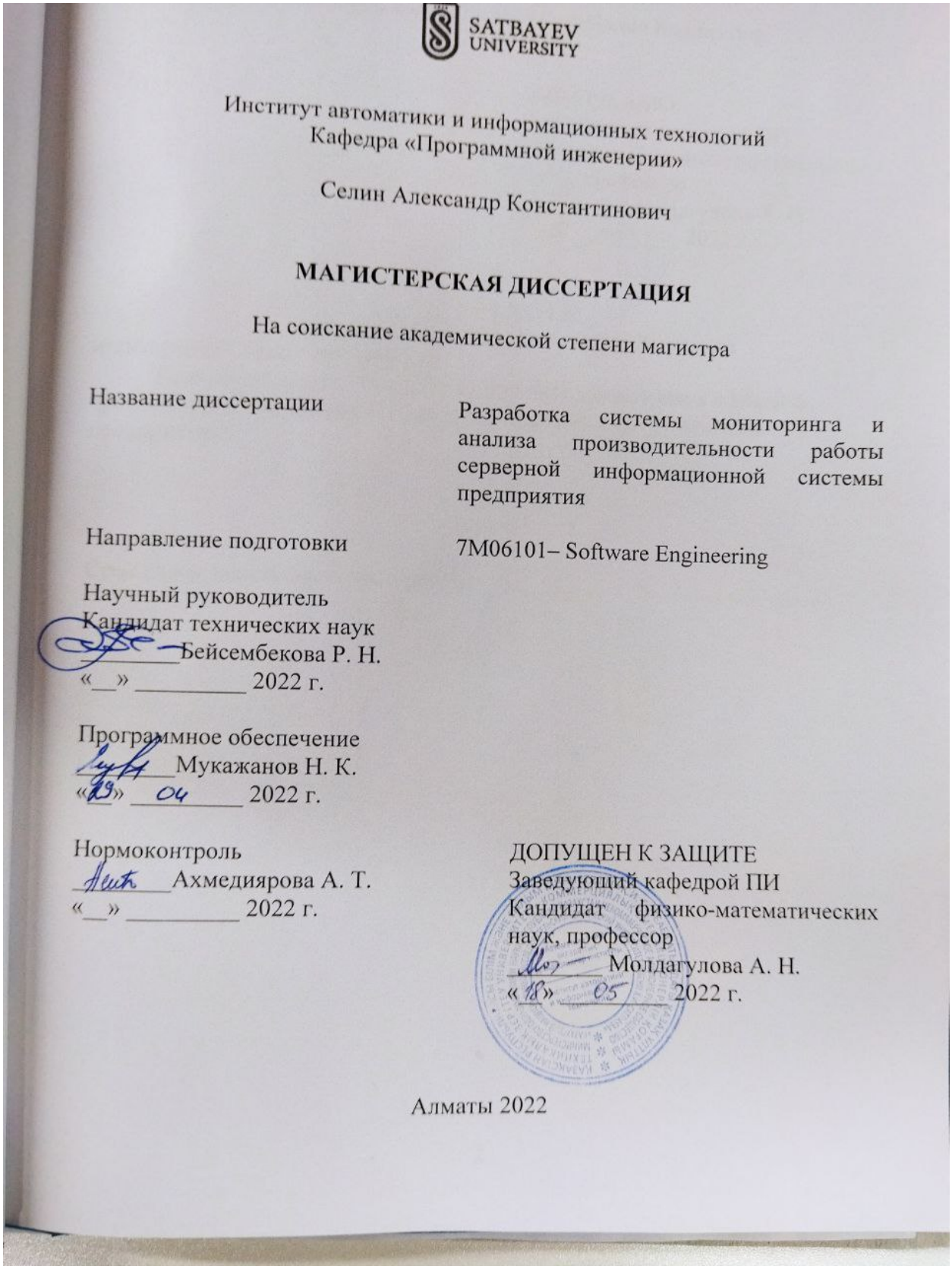
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

САТБАЕВ УНИВЕРСИТЕТІ



SATBAYEV UNIVERSITY

Институт автоматики и информационных технологий
Кафедра «Программной инженерии»

Селин Александр Константинович

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

На соискание академической степени магистра

Название диссертации — Разработка системы мониторинга и анализа производительности работы серверной информационной системы предприятия

Направление подготовки — 7М06101– Software Engineering

Научный руководитель
Кандидат технических наук
_____Бейсембекова Р. Н.
«__» _____ 2022 г.

Программное обеспечение
_____Мукажанов Н. К.
«23» 04 2022 г.

Нормоконтроль
_____Ахмедиярова А. Т.
«__» _____ 2022 г.

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ПИ
Кандидат физико-математических наук, профессор
_____ Молдагулова А. Н.
«18» 05 2022 г.

Алматы 2022

Институт автоматики и информационных технологий

Кафедра «Программной инженерии»

Специальность: 7М06101– Software Engineering

УТВЕРЖДАЮ
Заведующий кафедрой ПИ
Кандидат физико-математических наук, профессор
_____ Молдагулова А. Н.
«__» _____ 2022 г.

ЗАДАНИЕ
на выполнение магистерской диссертации

магистранту Селину Александру

Тема диссертации: «Разработка системы мониторинга и анализа производительности работы серверной информационной системы предприятия»

Срок сдачи законченной диссертации «__» _____

ГРАФИК

подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Раздел 1. Исследовательская предпроектная работа		
Раздел 2. Реализация		
Раздел 3. Анализ и мониторинг продуктивной системы		

Консультации по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант, (уч. степень, звание)	Сроки	Подпись
Нормоконтроль			

Дата выдачи задания "____"_____ 2022 г.

Заведующий кафедрой _____Молдагулова А. Н.

Научный руководитель _____ Бейсембекова Р. Н.

Задание принял к исполнению магистрант _____ Селин А. К.

Дата "_____"_____ 2022 г.

АННОТАЦИЯ

В настоящее время неотъемлемой частью обслуживания любой информационной системы является мониторинг и анализ её производительности. Важной частью этого процесса является автоматизация и визуализация полученных данных.

Исследовательская часть данной работы посвящена изучению вариантов визуализации метрических данных, собранных для мониторинга и анализа загруженности информационной системы. В качестве программ визуализации были выбраны Zabbix, PRTG и Grafana. Исследование представляет собой краткий сравнительный анализ выбранных программ.

Практическая часть – это непосредственная разработка ядра программного обеспечения, которое будет предоставлять данные по мониторингу производительности в виде метрик для визуализации. Основным новшеством является то, что на одном графике будут предоставлены данные и по удовлетворению пользователей от работы информационной системы, и по загруженности самой информационной системы.

В заключительной части разобраны несколько «кейсов» по улучшению производительности на продуктивном контуре, благодаря мониторингу и анализу данных, полученных в результате работы разработанного инструмента.

АҢДАТПА

Қазіргі уақытта кез келген ақпараттық жүйеге қызмет көрсетудің құрамдас бөлігі оның жұмысын бақылау және талдау болып табылады. Бұл процестің маңызды бөлігі алынған мәліметтерді автоматтандыру және визуализациялау болып табылады.

Бұл жұмыстың зерттеу бөлімі ақпараттық жүйенің жұмыс жүктемесін бақылау және талдау үшін жиналған метрикалық деректерді визуализациялау нұсқаларын зерттеуге арналған. Визуализация бағдарламалары ретінде Zabbix, PRTG және Grafana таңдалды. Зерттеу таңдалған бағдарламалардың қысқаша салыстырмалы талдауы болып табылады.

Практикалық бөлігі бағдарламалық қамтамасыз етудің өзегін тікелей әзірлеу болып табылады, ол визуализация үшін метрика түріндегі өнімділікті бақылау деректерін қамтамасыз етеді. Негізгі жаңалық – бір график ақпараттық жүйенің жұмысына пайдаланушының қанағаттануы туралы да, ақпараттық жүйенің жұмыс көлемі туралы да деректер береді.

Қорытынды бөлімде әзірленген құралдың нәтижесінде алынған деректерді бақылау және талдау арқасында өнімділік тізбегіндегі өнімділікті арттыру үшін бірнеше «жағдайлар» талданады.

SUMMARY

Currently, an integral part of the maintenance of any information system is the monitoring and analysis of its performance. An important part of this process is the automation and visualization of the received data.

The research part of this work is devoted to the study of options for visualizing metric data collected for monitoring and analyzing the workload of an information system. Zabbix, PRTG and Grafana were chosen as visualization programs. The study is a brief comparative analysis of the selected programs.

The practical part is the direct development of the software core, which will provide performance monitoring data in the form of metrics for visualization. The main innovation is that one graph will provide data on both user satisfaction from the operation of the information system and the workload of the information system itself.

In the final part, several “cases” are analyzed to improve performance on the productive circuit, thanks to the monitoring and analysis of data obtained as a result of the developed tool.

СОДЕРЖАНИЕ

Введение

Большинство предприятий в определённый момент своего жизненного цикла сталкиваются с проблемами производительности информационных систем. Больше пользователей, больше данных, больше требований к скорости отклика системы и множество других причин, из-за которых текущая производительность информационной системы перестаёт устраивать. Есть несколько вариантов решения такой проблемы – изменить информационную систему (ПО), поменять или добавить оборудование (железо), изменить и ПО, и железо. Такие способы обычно действенны, но негарантированно и, чаще всего, временно до следующего увеличения пользователей, данных и т.д.

Для того, чтобы «вылечить болезнь», а не устранить симптомы, необходимо произвести анализ загруженности оборудования, выявить узкие места в самой информационной системе и обсуживающей эту систему СУБД и произвести точечные настройки и апгрейд. А после произведенных действий настроить мониторинг информационной системы на значимые показатели, что позволит заранее до появления новых «симптомов» принять соответствующие меры.

Целью данной магистерской диссертации является разработка инструмента, системы инструментов, предоставляющих сервис по автоматизированному анализу и мониторингу оборудования, клиентской и серверной частей информационной системы и, если таковая используется, обслуживающую информационную систему СУБД. Для упрощения будет принято за постулат то, что каждая информационная система имеет трёхзвенную архитектуру – клиентская часть, сервер приложения и сервер СУБД. Также для информационной системы обязательным критерием для возможности проведения полноценного анализа будет то, что время начала и время окончания всех ключевых операций будет логироваться в специальном формате и специальном файле. Это условие необходимо для понимания качественного улучшения производительности информационной системы после проведенного анализа и выполненных рекомендованных на основе этого анализа работ по улучшению производительности.

В ходе реализации данной цели были поставлены и реализованы следующие задачи:

- написание скриптов по запуску сбора счетчиков оборудования для операционных систем семейства Windows[1] и Linux (CentOS)[2];
- написание скриптов по сбору данных по запросам для СУБД MS SQL[3] и PostgreSQL[4];

1 [6] Джон Мак-Кейб «Введение в Windows Server 2016»

2 [3] Алексей Береснев «Администрирование GNU/Linux с нуля»

3 [2] Александр Бондарь «Microsoft SQL Server 2014»

4 [12] Моргунов Е. П. «PostgreSQL. Основы языка SQL: учеб. пособие»

- написание скриптов по сбору данных по технологическому журналу информационной системы на платформе 1С:Предприятие[5];

- создание инструмента для хранения и анализа собранных этими скриптами данных (включая данные по работе ключевых операций);

- создание тестового стенда и приближенных к реальной работе тестовых сценариев для сбора информации по производительности, не мешая работе реальных пользователей.

Для практической наглядности объектом исследования будет реальное предприятие – АО «Технодом Operator», а предметом исследования – информационная система «1С:Управление производственным предприятием».

Научный аппарат диссертации

5 [9] Е. Филлипов «Настольная книга 1С:Эксперта по технологическим вопросам»

Тема

Тема данной магистерской диссертации: «Разработка системы мониторинга и анализа производительности работы серверной информационной системы предприятия»

Объект

Объектом исследования данной магистерской диссертации является работа серверной и клиентской частей информационной системы «1С:Управление производственным предприятием» АО «Технодом Operator»

Предмет

Предметом исследования данной магистерской диссертации является производительность работы серверной и клиентской частей информационной системы «1С:Управление производственным предприятием» АО «Технодом Operator»

Цель

Цель исследования данной магистерской диссертации – разработать инструмент (систему инструментов) для мониторинга и анализа производительности работы серверной и клиентской частей информационной системы

Задачи

Задачи, которые необходимо решить в рамках данной магистерской диссертации:

– написание скриптов по запуску сбора счетчиков оборудования для операционных систем семейства Windows и Linux (CentOS);

– написание скриптов по сбору данных по запросам для СУБД MS SQL и PostgreSQL;

– написание скриптов по сбору данных по технологическому журналу информационной системы на платформе 1С:Предприятие;

– создание инструмента для хранения и анализа собранных этими скриптами данных (включая данные по работе ключевых операций);

– создание тестового стенда и приближенных к реальной работе тестовых сценариев для сбора информации по производительности, не мешая работе реальных пользователей

Гипотеза

Во всякой системе есть узкое место «бутылочное горлышко» (одно или несколько), которое является основным источником сильного снижения производительности во время нагрузки оборудования. Мониторинг и анализ производительности работы серверной и клиентской частей информационной системы на постоянной основе позволят выявить и устранить узкие места системы путём переписания запросов, добавления

индексов в СУБД либо перенастройки регламентных заданий, создающих нагрузку на систему

Новизна

Исследование подобных инструментов показало[6], что есть необходимость в создании нового, включающего в себя возможности сразу нескольких существующих. Например, для более глубокого анализа на график работы оборудования, показывающего производительность работы, необходимо наложить график индекса производительности пользователей, чтобы увидеть, как загруженность оборудования влияет на комфортность работы пользователей. То есть, прямую зависимость «производительность информационной системы – индекс комфортности работы». В этом заключается новизна данной магистерской диссертации

Методы

Логирование загруженности оборудования. Логирование работы СУБД. Логирование индекса производительности пользователей. Выборка и хранение информации из лог-файлов для мониторинга и анализа. Автоматический анализ. Автоматическое оповещение администраторов информационной системы по электронной почте при достижении нежелательных показателей

Теоретическая значимость

Разработка системы мониторинга и анализа производительности работы серверной и клиентской частей информационной системы позволит улучшить понимание методов и способов нахождения узких мест работы информационных систем, создаст теоретические предпосылки для проектирования универсальных рекомендаций и создания автоматических анализаторов

Практическая значимость

Разработка системы мониторинга и анализа производительности работы серверной и клиентской частей информационной системы позволит улучшить производительность информационной системы. А значит повысить комфортность работы пользователей конкретной информационной системы «1С:Управление производственным предприятием» АО «Технодом Operator»

Исследовательская предпроектная работа

6 [1] А. Морозов, В. Федоров, А. Асатрян, А. Голиков, Д. Соломатин «Методическое пособие по эксплуатации крупных информационных систем на платформе «1С:Предприятие 8»»

Для проведения более предметной исследовательской работы решено было сузить задачи, рассматриваемые в данной диссертации, до требований конкретного заказчика. А именно – компании АО «Технодом Operator» с разветвленной сетью серверов и рабочих станций, на которых установлены операционные системы семейства Windows. То есть, требования следующие:

1. Операционная система: Windows Server 2012 или Windows 10
2. СУБД: MS SQL 2016
3. Клиентское приложение: «1С:Предприятие 8.3»

На рисунке №1 представлена блок-схема итогового программного продукта

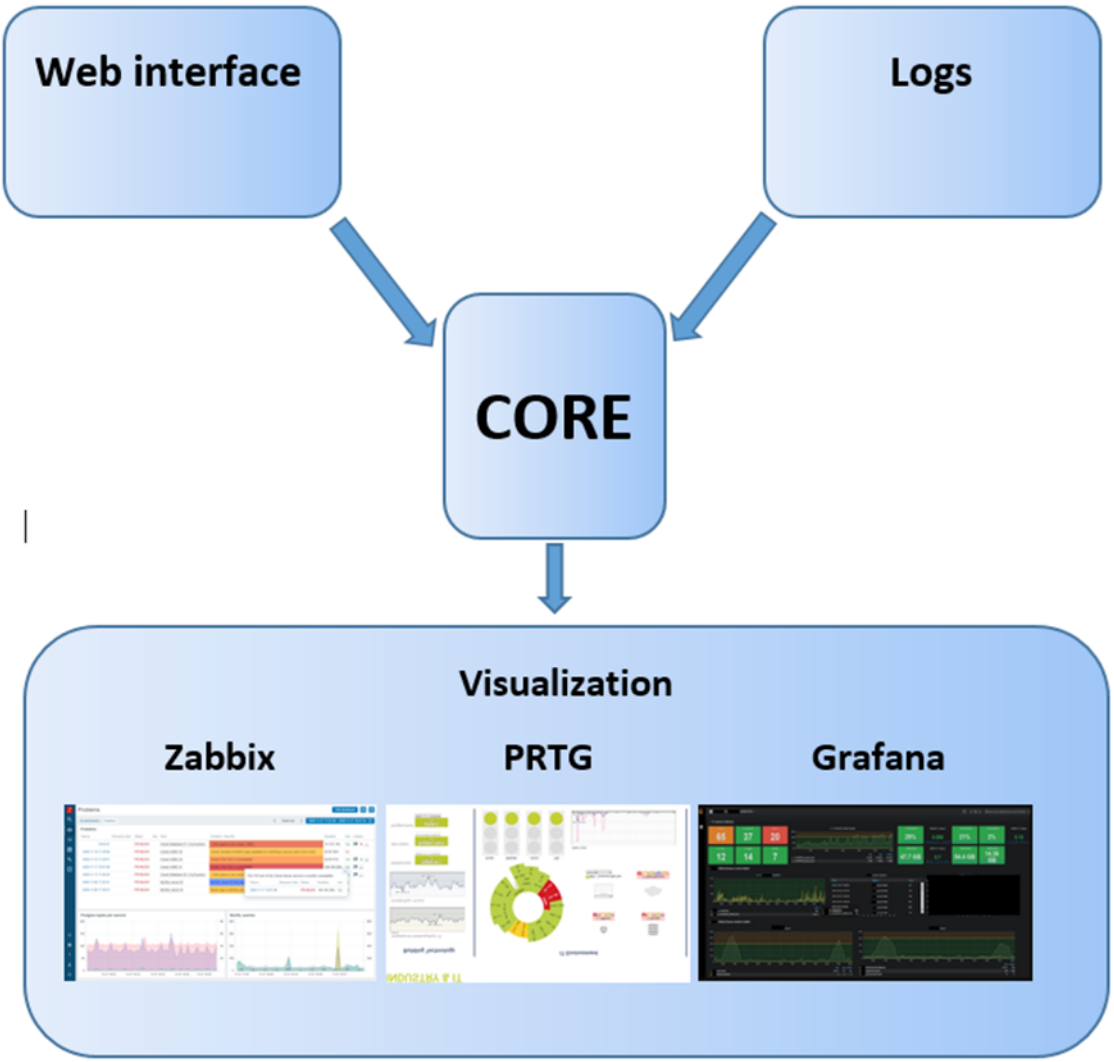


Рисунок №1. Блок-схема ПО

На этапе разработки архитектуры в качестве визуализаторов полученной метрической информации, как потенциальные кандидаты, выбраны Zabbix, PRTG или Grafana.

Ядро программы (Core) и веб-интерфейс для настроек мониторинга и анализа будут написаны на языке программирования Python.

Сбор информации будет осуществляться из соответствующих лог-файлов, которые будут собираться после настройки программы

Блок сборки информации представлена на рисунке №2

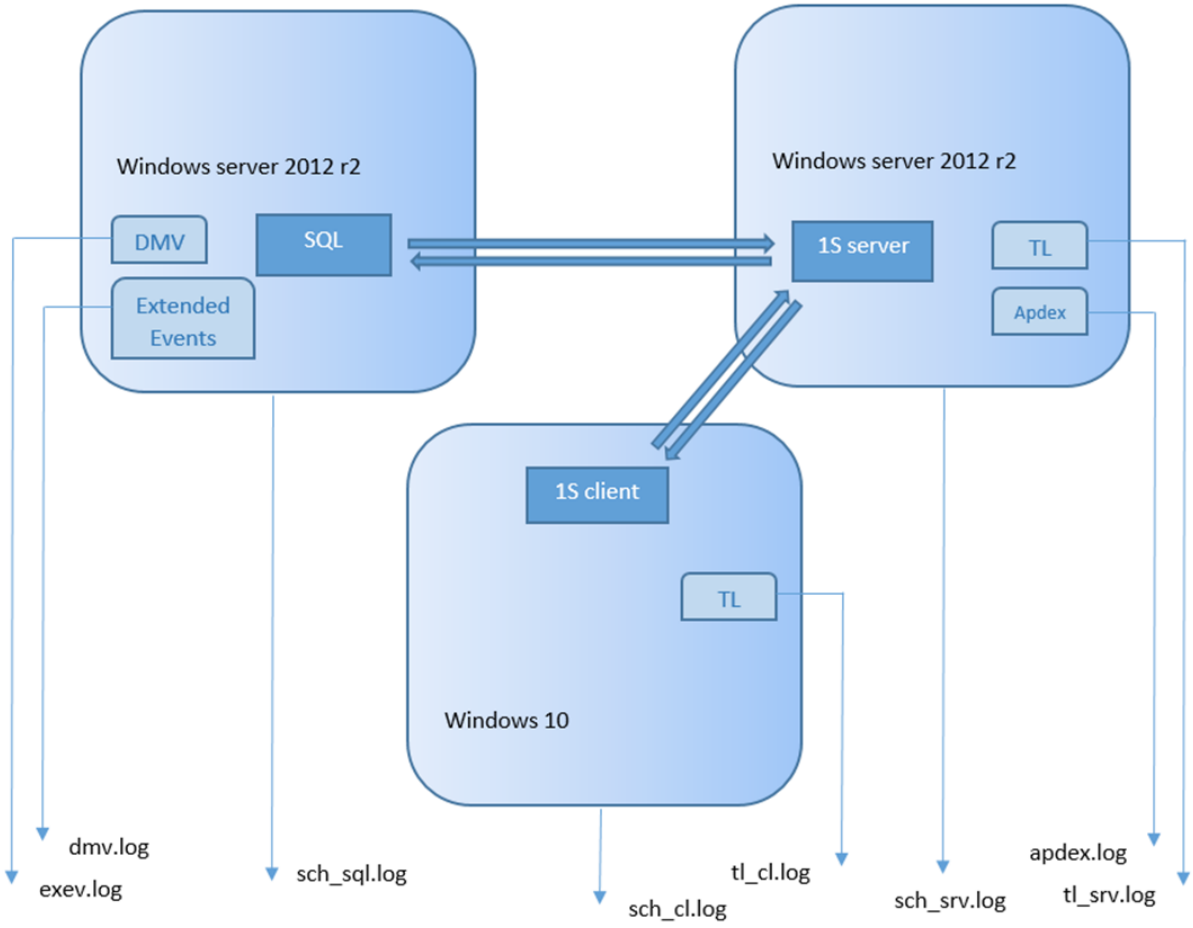


Рисунок №2. Блок-схема блока сборки информации

Описание лог-файлов представлено в таблице 1.

Таблица 1.

Описание лог-файлов

№	Log name	Log description
1	dmv.log	MS SQL Dynamic Management Views – statistics on query execution time and resources used
2	exev.log	MS SQL Extended Events – monitor for various query execution metrics
3	apdex.log	Apdex – Application Performance Index
4	tl_srv.log	Technological log on the application server. Information on exceptions, on requests, on locks
5	tl_cl.log	Technology log on the client. Information on exceptions, on requests, on locks
6	sch_srv.log	Hardware performance counter on the application server
7	sch_cl.log	Hardware performance counter on the client
8	sch_sql.log	Hardware performance counter on SQL server, SQL process counter

Для проведения предпроектной исследовательской работы были выбраны следующие программные продукты:

1. Zabbix[7]
2. PRTG[8]
3. ЦУП[9]
4. ЦКК[10]
5. Анализатор ТЖ[11]
6. Сервисы Гилёва[12]
7. Grafana[13]

Идея состоит в том, чтоб после исследования появилось более полное представление о требованиях к разработке итогового программного продукта, исходя достоинств исследуемых программных продуктов, и чтоб в итоговом программном продукте не оказалось недостатков исследуемых программных продуктов.

Zabbix

Zabbix – это свободно распространяемое ПО для мониторинга состояния компьютерных сетей, серверов и другого оборудования, написанная Алексеем Владышевым.

На сегодняшний день входит в тройку самых популярных программ для мониторинга. Состоит из zabbix сервера и zabbix агента. Сервер устанавливается исключительно на операционных системах семейства Linux. Агент может быть установлен как на Windows, так и на Linux. Для проведения исследовательской работы сервер будет установлен на Ubuntu, агент – на Windows Server 2012.

По итогу исследований получен вывод, что в качестве аналога для предмета исследования данной диссертации zabbix подходит частично. Zabbix позволяет мониторить оборудование и отдельные запущенные процессы (например процесс ms sql) по различным характеристикам. Однако, для того, чтобы использовать эту систему в качестве анализатора, необходимо написать этот анализатор отдельно, а данные уже выгружать в систему Zabbix. То есть, в части мониторинга и визуализатора эта система может заменить программный продукт, который будет являться итогом данной диссертации. Но в части анализа полученных из лог-файлов данных – нет. Само получение лог-файлов также придется настраивать не в системе Zabbix.

---

7 [18] https://www.zabbix.com/ru/

8 [19] https://www.ru.paessler.com/

9 [20] https://v8.1c.ru/tekhnologii/tekhnologii-krupnykh-vnedreniy/korporativnyy-instrumentalnyy-paket/tsentr-upravleniya-proizvoditelnostyu

10 [21]https://v8.1c.ru/tekhnologii/tekhnologii-krupnykh-vnedreniy/korporativnyy-instrumentalnyy-paket/tsentr-kontrolya-kachestva

11 [22] https://infostart.ru/1c/articles/1040073

12 [23] http://www.gilev.ru

13 [24] https://grafana.com

На рисунке №3 можно увидеть, как выглядит Zabbix.

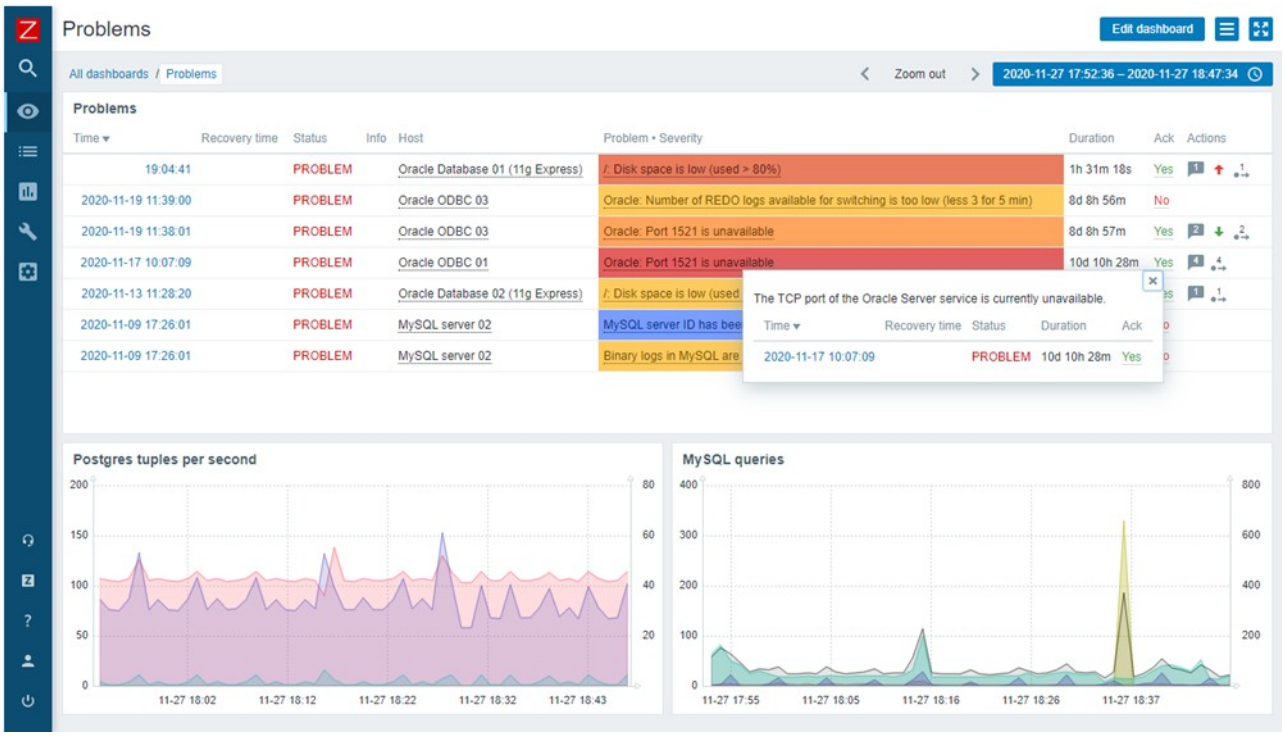


Рисунок №3. Пример Zabbix

PRTG

PRTG (Paessler Router Traffic Grapher) – условно-бесплатное ПО. После 30 дней ограничивается количество сенсоров. Предназначено для мониторинга сети, но есть возможность мониторить нагрузку оборудования у конкретных рабочих станций в сети. Устанавливается только в семействе операционных систем Windows, что хорошо для поставленной задачи данной диссертации.

Возможности программы:

- сбор информации о потоках данных, проходящих через конкретные устройства, с сохранением её в базе данных программы;
- сбор данных через SNMP, Netflow и множество других протоколов;
- просмотр статистики в базе данных в виде графиков и таблиц;
- статистка переданных пакетов данных и времени пинга;
- просмотр результатов в режиме реального времени или за определенный промежуток времени в прошлом на разных устройствах;
- сбор данных о нагрузке на подсистемы памяти и процессор.

В компании АО «Технодом Operator» уже был опыт использования PRTG. Этот программный продукт хорошо зарекомендовал себя, однако он всё же платный, поэтому от него отказались. В качестве использования в целях данной диссертации он не подходит по тем же причинам, что и Zabbix.

Нет собственных средств для проведения анализа и для поиска узких мест информационной системы. PRTG – это мониторинг. Для анализа нужно будет дописывать целый блок, данные которого уже отправлять в PRTG

Как выглядит PRTG визуально можно увидеть на рисунке №4

Рисунок №4. Пример PRTG

ЦУП

Центр управления производительностью (ЦУП) – инструмент, разработанный компанией «1С», для мониторинга и анализа производительности клиент-серверных информационных систем на платформе «1С:Предприятие 8».

Практические задачи, которые могут быть решены при помощи ЦУП:

1. Анализ производительности многопользовательской информационной системы:
   - вопрос проблемы производительности;
   - вопрос повышения производительности.
2. Сбор и хранение информации о динамике производительности системы:
   - изменение производительности системы с течением времени;
   - изменение производительности системы при внесении каких-либо новшеств.
3. Поиск и анализ узких мест конфигурации. Получение детальной технической информации обо всех проблемах производительности, имеющихся в системе, с целью дальнейшей оптимизации:

- проблемы производительности имеются в системе и насколько они серьезны;
- какие проблемы следует решать в первую очередь;
- в чем заключается каждая проблема;
- какие строки кода конфигурации нужно оптимизировать, для того чтобы решить проблему.

Главным недостатком этого инструмента является то, что он платный. Также он требует мощного оборудования для своей работы. Визуальная составляющая тоже заставляет желать лучшего. Однако, этот инструмент уже качественно может анализировать производительность. На рисунке №5 и рисунке №6 показан пример интерфейса ЦУП.

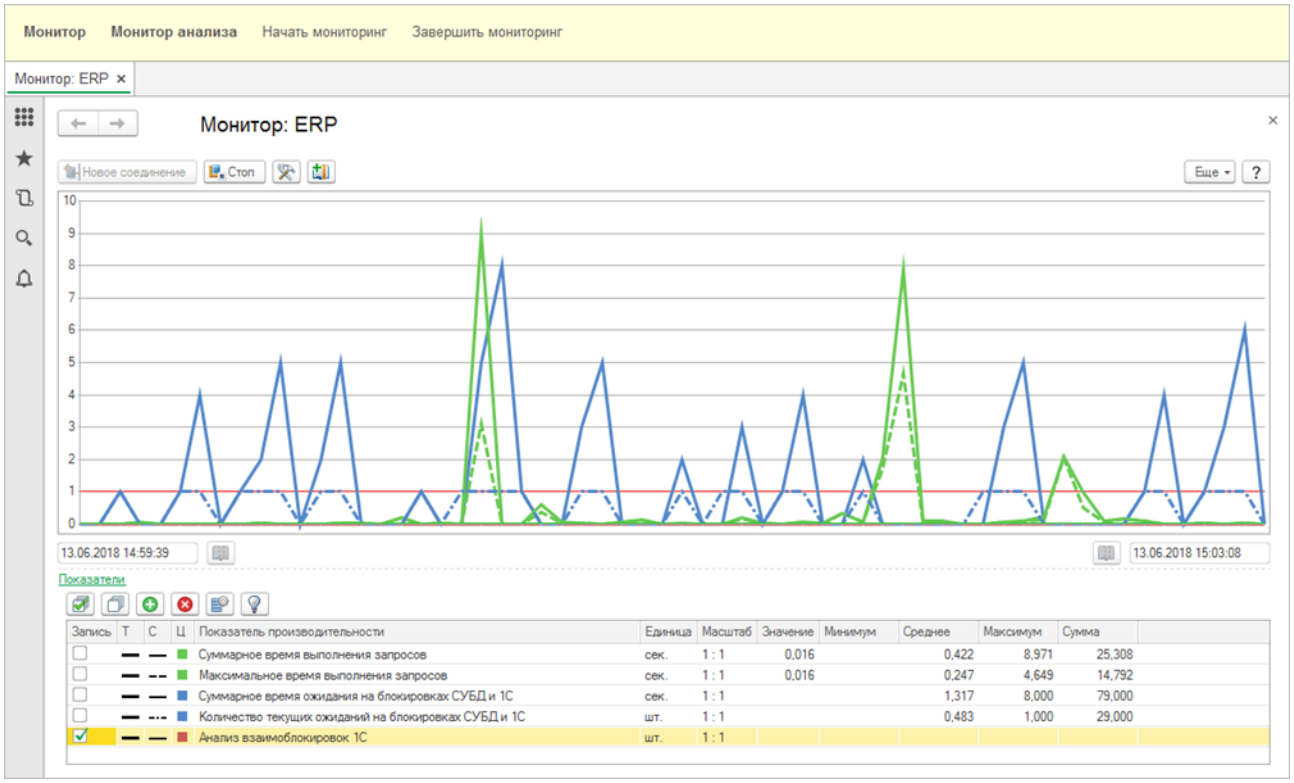


Рисунок №5. Пример ЦУП. Монитор

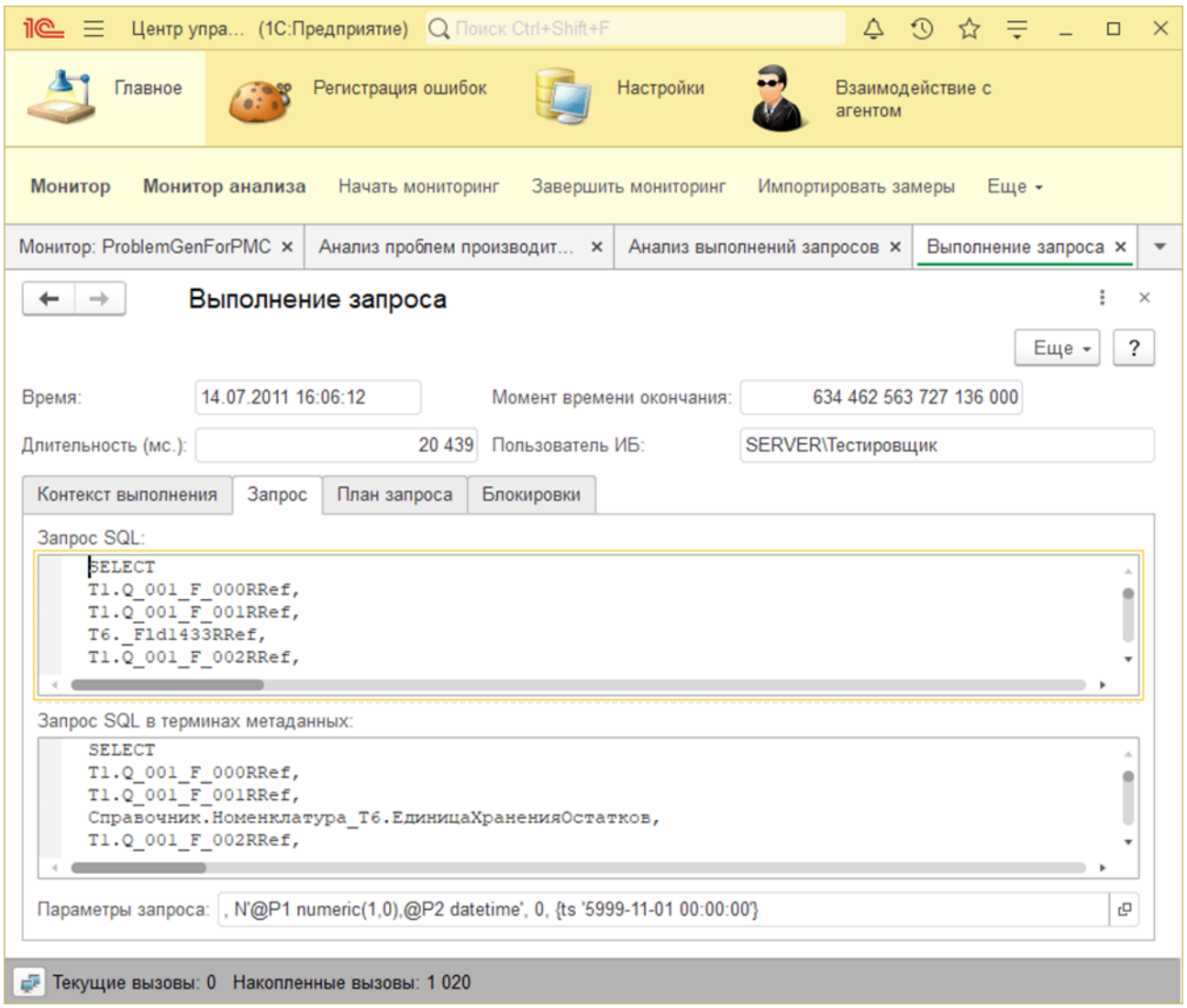


Рисунок №6. Пример ЦУП. Анализ

ЦКК

Центр контроля качества (ЦКК) – это инструмент, разработанный специалистами компании «1С» на базе платформы «1С:Предприятие 8» и отдельного приложения «Агент КИП» на java, предназначенный для обеспечения и повышения технологического качества работы информационных систем на базе платформы «1С:Предприятие 8».

ЦКК выполняет следующие функции:

1. Мониторинг состояния контролируемой информационной системы;
2. Отображение графиков и диаграмм состояния информационной системы;
3. Своевременное обнаружение и анализ проблем;
4. Хранение и обработка технологических данных о состоянии контролируемой информационной системы;
5. Извещение ответственных о возникновении ситуаций, требующих вмешательства;

6. Использование для нагрузочных, подготовительных и других стендов на базе платформы «1С:Предприятие 8», сбора статистики об их работе

Это мощный инструмент. Но он также, как и ЦУП, платный, требует хорошее оборудование и визуализация не на высшем уровне. Также он обладает не совсем теми функциями, которые нужны. Он больше анализирует технологическое качество, чем производительность.
Пример показан на рисунке №7

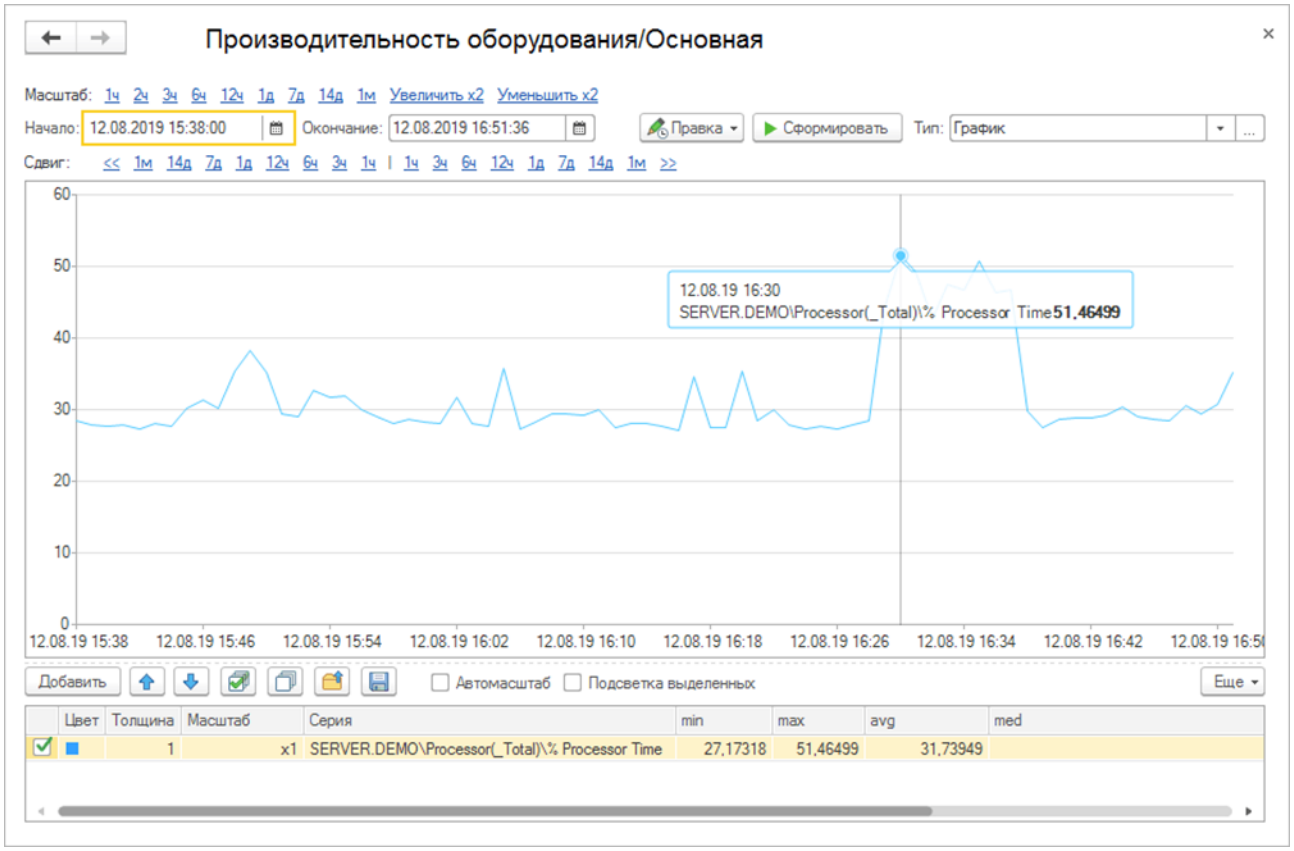


Рисунок №7. Пример ЦКК

Анализатор ТЖ

Конфигурация, встраиваемая, как отдельный блок, в продуктивную конфигурацию и позволяющая производить мониторинг и анализ производительности. Это open source проект в задачи которого входит мониторинг проблем производительности
Позволяет:

1. Загружать данные технологического журнала 1С;
2. Загружать данные счетчиков производительности серверов windows и MS SQL;
3. Загружать данные из произвольных источников по средствам плагинов: zabbix api;

4. Загружать данные из сервиса 1С RAS;
5. Отображать информацию в наглядном виде в различных автоматизированных рабочих местах (обработках);
6. Отображать информацию в журнале в форме таблицы;
7. Выполнять обработку входных данных и выдавать экспертное заключение с применением искусственного интеллекта (блок нечеткой логики);
8. Выполнять оповещения по @, SMS, Skype и др.;
9. Анализ проблем производительности с помощью нейронных сетей;
10. Автоклассификация ошибок технологического журнала 1С.

Как у большинства open source проектов, у этого программного продукта нет поддержки и того, кто будет отвечать за нестабильность работы. Это такой проект, на который можно посматривать, разрабатывая свой.
Один из его отчетов показан на рисунке №8.

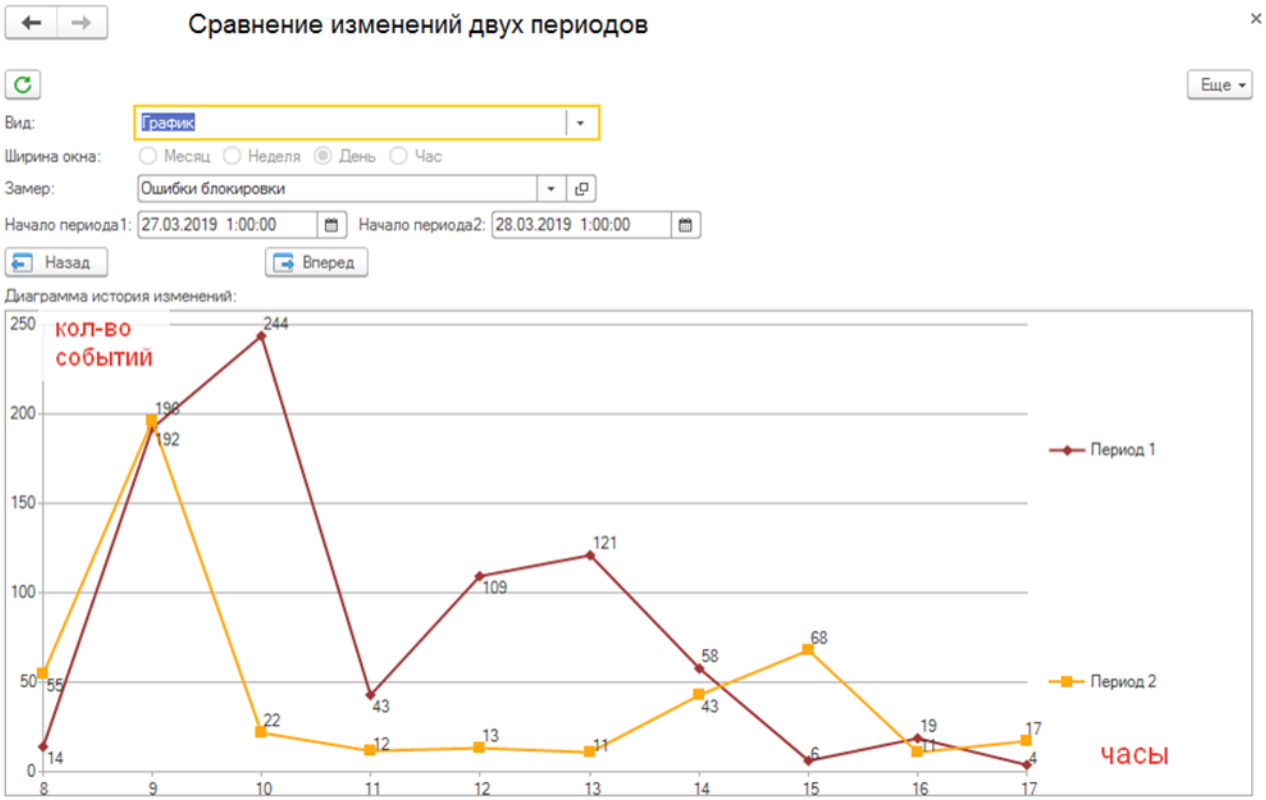


Рисунок №8. Пример ЦУП. Монитор

Сервисы Гилёва

Это набор платных услуг по аудиту производительности информационной системы, предоставляемый командой под руководством эксперта 1С Вячеслава Гилёва. Варианты услуг различны. Команда может подключиться к вам удаленно, поработать со специалистами компании, собрать какие-то данные и дать своё заключение. Либо можно подключить

их средства сбора данных в свою информационную систему и видеть результат мониторинга и анализа в их облаке в личном кабинете через браузер.

Чем интересны эти сервисы для данной диссертации – это возможностью перенять методы работы команды Гилёва, «подсмотреть» какими могут быть результаты, перенять их способы анализа и повышения производительности. В это исследование включен пункт про сервисы Гилёва исключительно для повышения кругозора в сфере инструментов для аудита и улучшения производительности. Сервисы Гилёва не рассматриваются как аналог или замена продукта, который получится в итоге реализации данной диссертации. Но для полноты картины подходит, как нельзя лучше.

Вот что они пишут о себе, и что будет необходимо реализовать в конечном продукте данной диссертации:

1. Узкая специализация по вопросам производительности 1С:Предприятие 8
2. Делаем нашу работу ЛУЧШЕ других
3. Именно нашим тестом замеряют «скорость 1С»
4. Мы обучили наибольшее количество технических специалистов решать вопросы производительности
5. На основе наших публичных и не публичных материалов строят свои методички все крупные вендоры на рынке
6. Поможем Вам правильно сформулировать задачи к исполнению
7. Соберем статистику производительности Вашей информационной системы
8. Сделаем прозрачным отслеживание состояние исследуемой проблемы
9. Организуем онлайн доступ к мониторингу по всему миру, хоть из самолета
10. Автоматизируем анализ сложных причин, вызывающих замедление

На рисунках №9 и №10 изображены скриншоты работы этих сервисов.

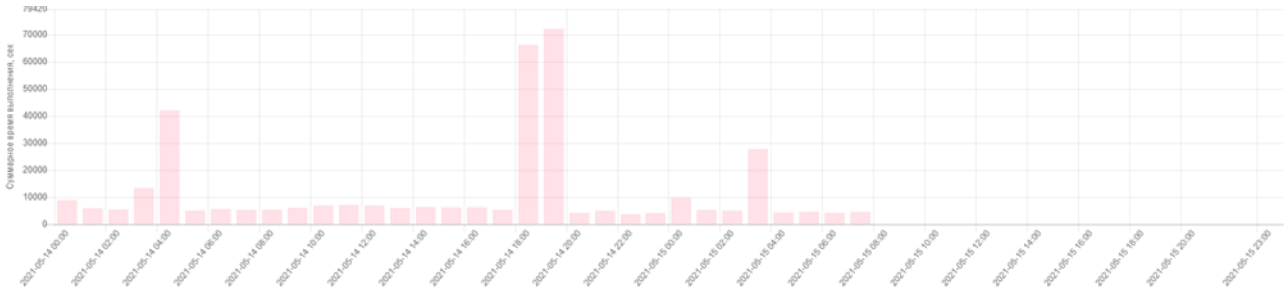


Рисунок №9. Пример отчета работы сервиса Гилёва

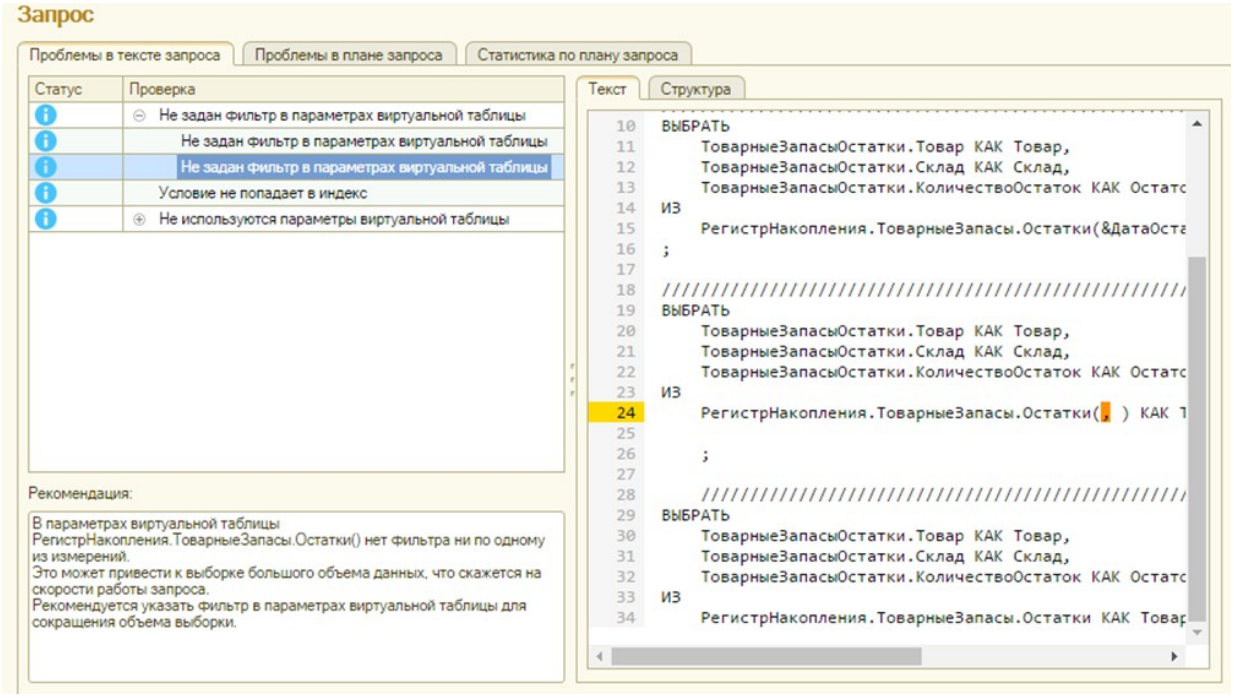


Рисунок №10. Пример анализа работы сервиса Гилёва

Grafana

Это последний из исследуемых продуктов. И он отличается тем, что сам не собирает никакие данные и производит анализов. Это удобный, красивый и бесплатный визуализатор уже собранных данных и метрик. Скорее всего, именно он и будет использоваться в продукте данной диссертации для «вывода» данных пользователю в виде дашбордов, графиков и диаграмм.
На рисунке №11 показано как это может выглядеть.

Рисунок №11. Пример Grafana

Вывод

На рынке платных и бесплатных программных продуктов есть множество различных мониторинговых систем и систем, производящих анализ работы операционных систем, СУБД и приложений. Рассмотрена только небольшая часть из них. Однако, это рассмотрение позволило сделать вывод, что все эти программные продуты либо мониторят, либо анализируют. А нужно и то, и другое

Подготовка к реализации проекта (установка ПО, дорожная карта)

Перед началом реализации проекта необходимо понять какие шаги нужно предпринять. Для этого была разработана дорожная карта проекта. Выглядит она следующим образом:

Дорожная карта:

1. MS SQL Extended Events
   - 1.1. Настроить вручную
   - 1.2. Настроить автоматически – скриптом
   - 1.3. Настроить автоматически – скриптом, запущенным при помощи python
   - 1.4. Обработать полученные лог файлы вручную
   - 1.5. Обработать полученные скриптом
   - 1.6. Обработать полученные скриптом, запущенным при помощи python
2. MS SQL Dynamic Management Views
   - 2.1. Собрать данные вручную
   - 2.2. Поместить данные вручную в соответствующий лог файл
   - 2.3. Собрать данные скриптом, запущенным при помощи python, в соответствующий лог файл
3. Grafana + Prometheus
   - 3.1. Развернуть на тестовом стенде Prometheus
   - 3.2. Развернуть на тестовом стенде Grafana
   - 3.3. Настроить в Grafana источник данных на базу Prometheus
   - 3.4. Получить тестовые графики в Grafana
4. Счетчики оборудования и SQL
   - 4.1. Настроить вручную
   - 4.2. Настроить автоматически – скриптом
   - 4.3. Настроить автоматически – скриптом, запущенным при помощи python
   - 4.4. Добавить данные по счетчикам в базу Prometheus
5. Технологический журнал
   - 5.1. Настроить вручную
   - 5.2. Настроить автоматически – скриптом, запущенным при помощи python
6. Анализ и мониторинг
   - 6.1. Разработать метрики для базы Prometheus из Extended Events, Dynamic Management Views, счетчиков, технологического журнала и данным апдекса
   - 6.2. Продумать, что будет аналитическими данными, а что мониторинговыми
   - 6.3. Продумать и разработать регламент, который будет передавать данные на постоянной основе в базу Prometheus
   - 6.4. Настроить удобные дашборды в Grafana для мониторинга

7. Веб-интерфейс для настроек
   7.1. Продумать веб-интерфейс для запуска автоматического сбора данных для анализа и мониторинга
   7.2. Реализовать при помощи python
8. Сборка и инсталляция
   8.1. Собрать ядро проекта
   8.2. Запустить выполнение мониторинга и анализа на тестовых данных
   8.3. Получить результаты. Проанализировать их. Сделать выводы

Последовательность шагов необязательная. Можно, например, в начале сделать различные «заготовки» – установить необходимое ПО, сделать шаблон веб-интерфейса, разработать различные части ядра программы. Это не помешает выполнению проекта. Главное, чтоб перед сборкой и инсталляцией все пункты дорожной карты были выполнены.

Установка Prometheus и Grafana

Для установки Prometheus и Grafana в виртуальной среде был создан сервер с операционной системой «Ubuntu 20.04.3 LTS» **vm-test-sea** (Virtual Machine – TEST – SElin Alexandr). С помощью ПО «MobaXterm» производилось управление этим сервером. На рисунке №12 скриншот этого ПО.

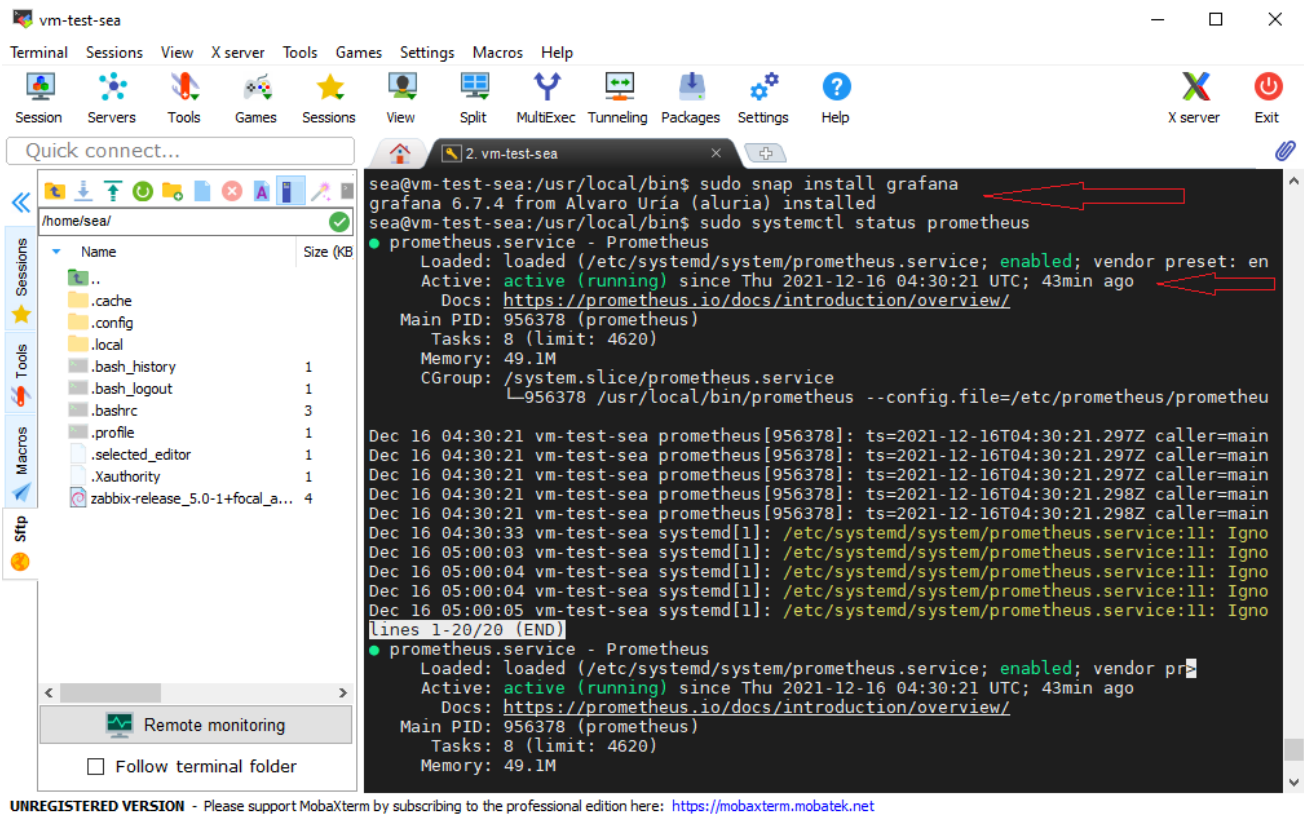


Рисунок №12. ПО «MobaXterm»

Опираясь на найденную в интернете инструкцию,[14] были развернуты Prometheus и Grafana. Так же были произведены настройки источника данных в Grafana на базу Prometheus и добавлен тестовый дашборд. На скриншотах видно, что через браузер можно управлять отображаемыми данными и адаптировать под себя их визуализацию.
На рисунке №13 показан веб-интерфейс развёрнутого Prometheus, а на рисунке №14 – тестовый дашборд в Grafana, настроенный на базу Prometheus.

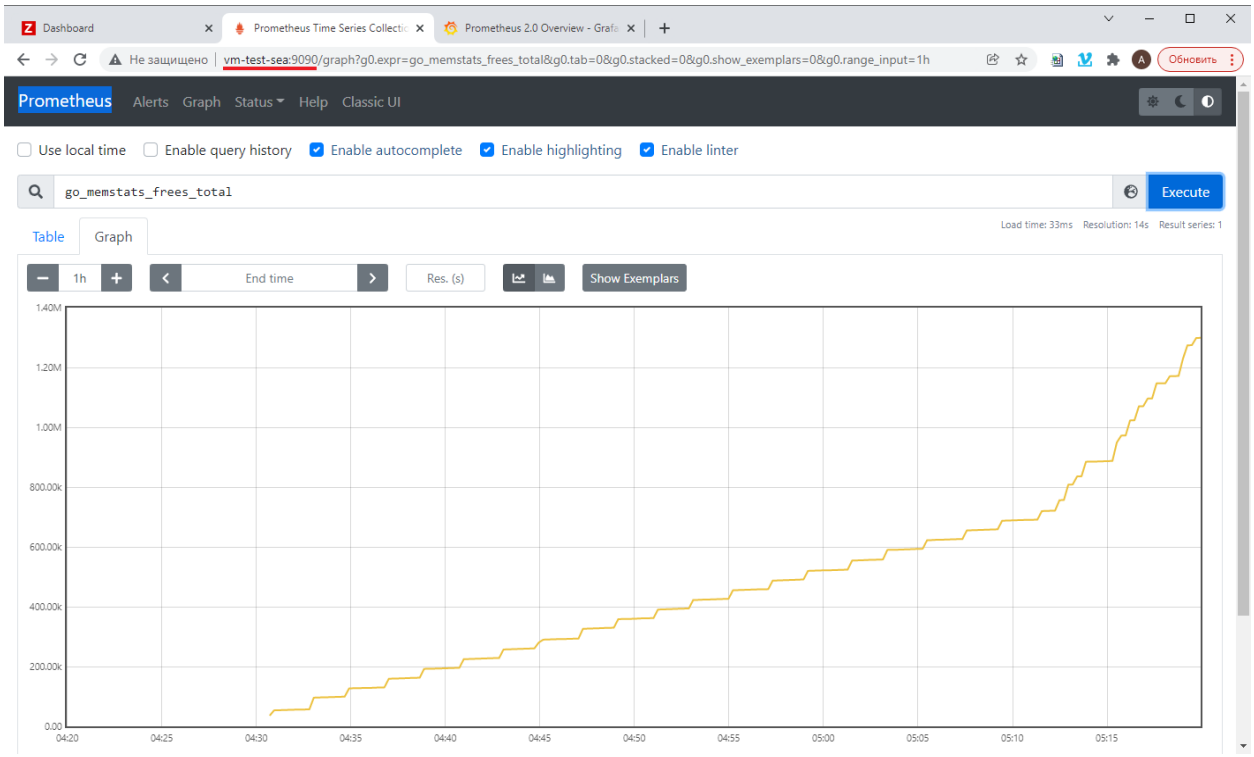


Рисунок №13. Веб-интерфейс Prometheus

14 [25] https://losst.ru/ustanovka-i-nastrojka-prometheus

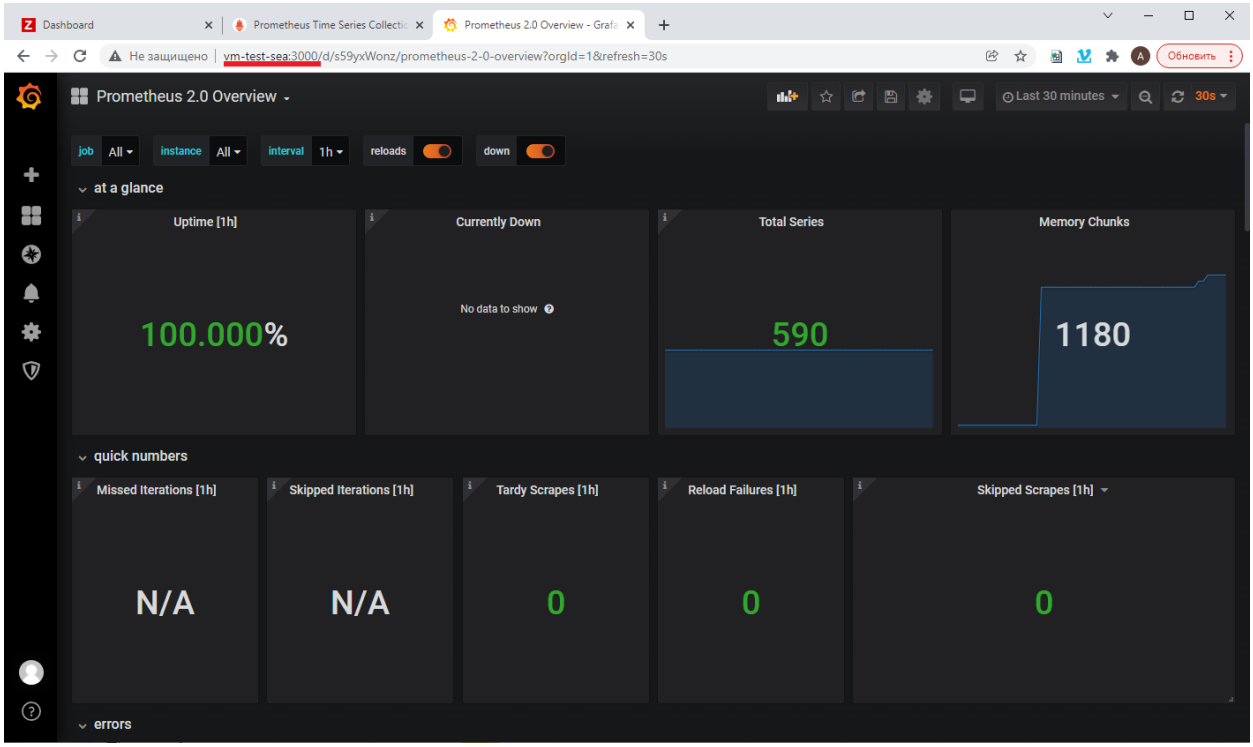


Рисунок №14. Тестовый дашборд в Grafana

Также для возможности расширения проекта и какого-то вероятного ответвления на сервере vm-test-sea был развернут Zabbix. Никаких настроек на нём не производилось. При необходимости Zabbix будет настроен и графики с дашбордов Grafana и Zabbix можно будет сравнить и выбрать более подходящий.

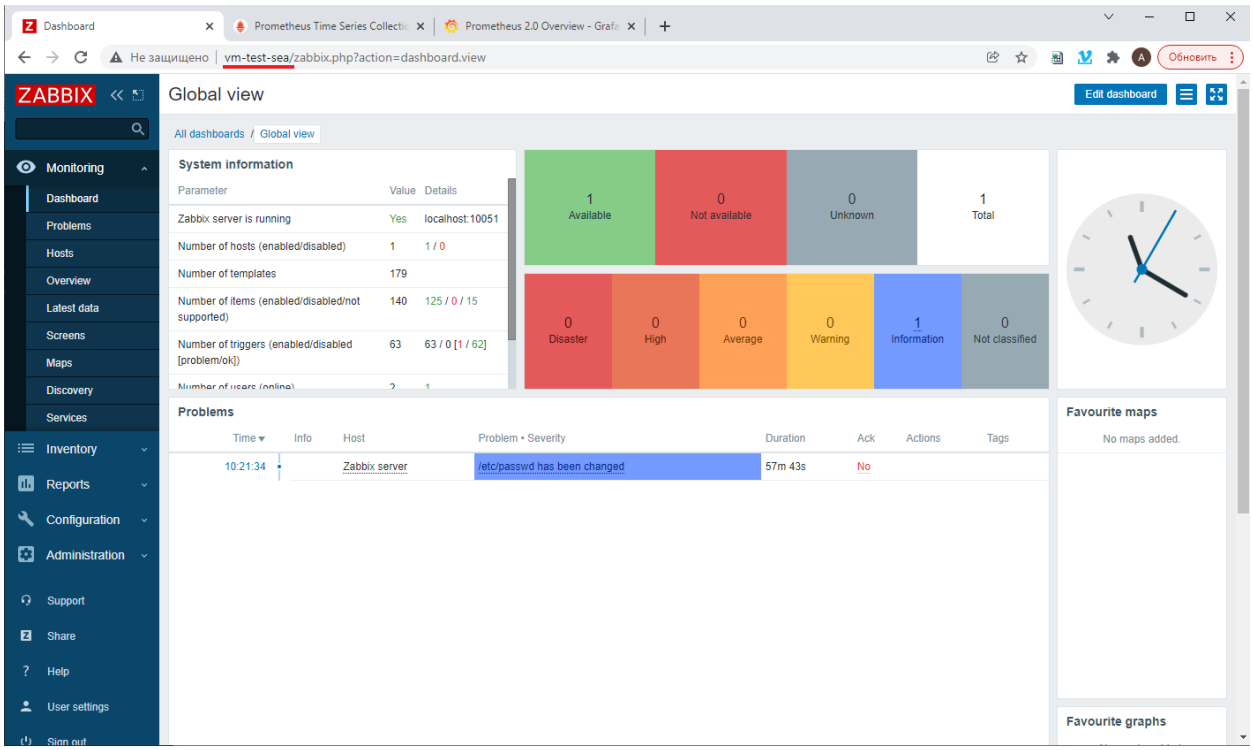


Рисунок №15. Тестовый дашборд в Zabbix

Веб-интерфейс для настроек

Реализацию веб-интерфейса было решено делать при помощи фреймворка «Flask» для python. Инструкция по его установке и быстрому старту была взята с официального сайта[15]. Применение этого фреймворка позволяет небольшому блоку кода на python показывать на веб-сервере (для этого надо в терминале выполнить несколько команд для запуска) веб-страницу для ввода настроек.
Вот блок кода:

```
from flask import Flask, render_template

app = Flask(__name__)


@app.route("/")
def index():
    return render_template('index.html')
```

Команды для запуска веб-сервера:
set FLASK_APP=server
set FLASK_ENV=development
flask run

На рисунке №16 показан скриншот с полученной веб-страницей.

15 [26] https://flask.palletsprojects.com/en/2.0.x/quickstart/

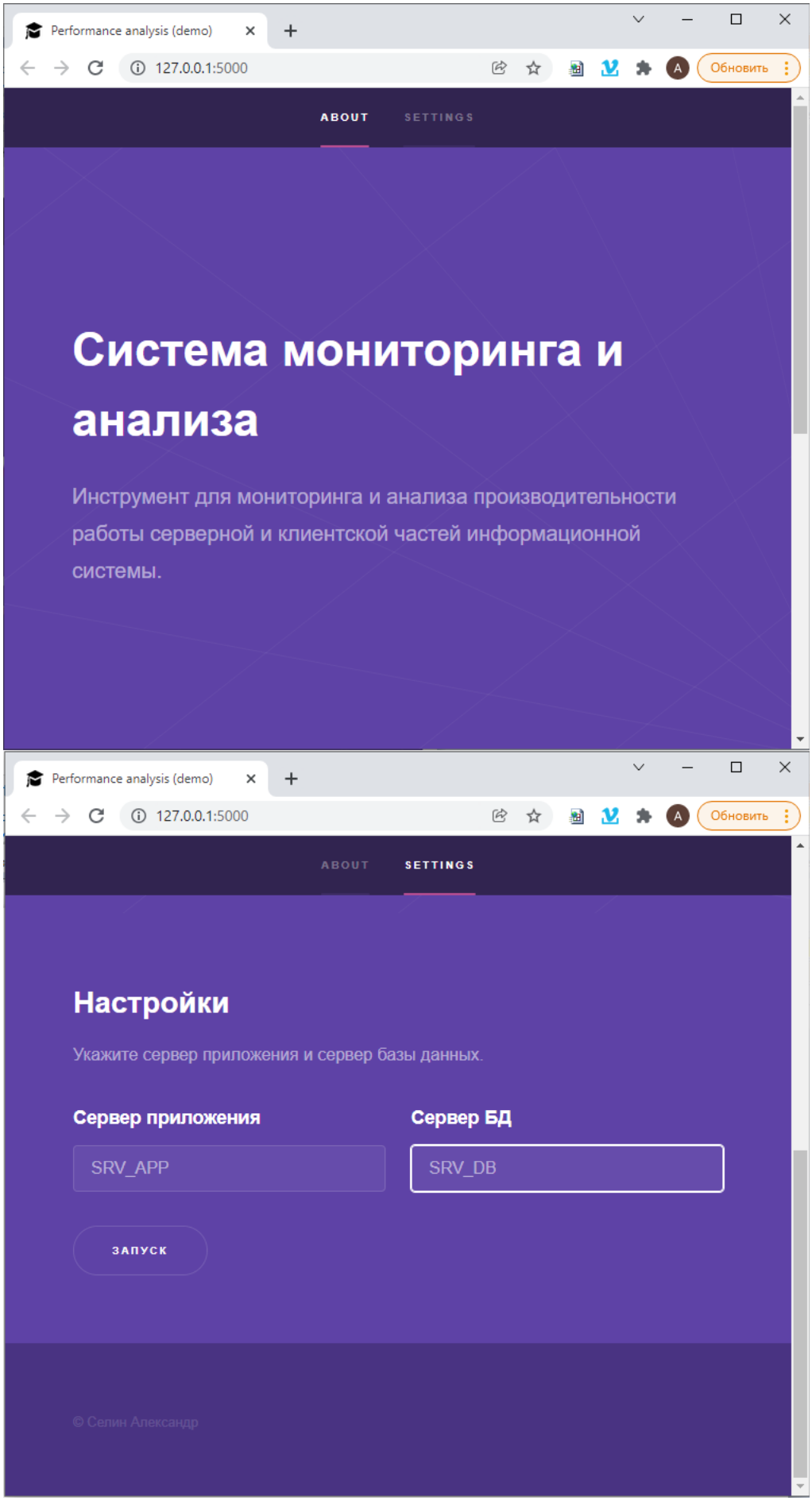


Рисунок №16. Веб страница для настроек

Скриншот (рисунок №17) из PyCharma с кодом страницы index.html и командами для запуска веб-сервера flask.

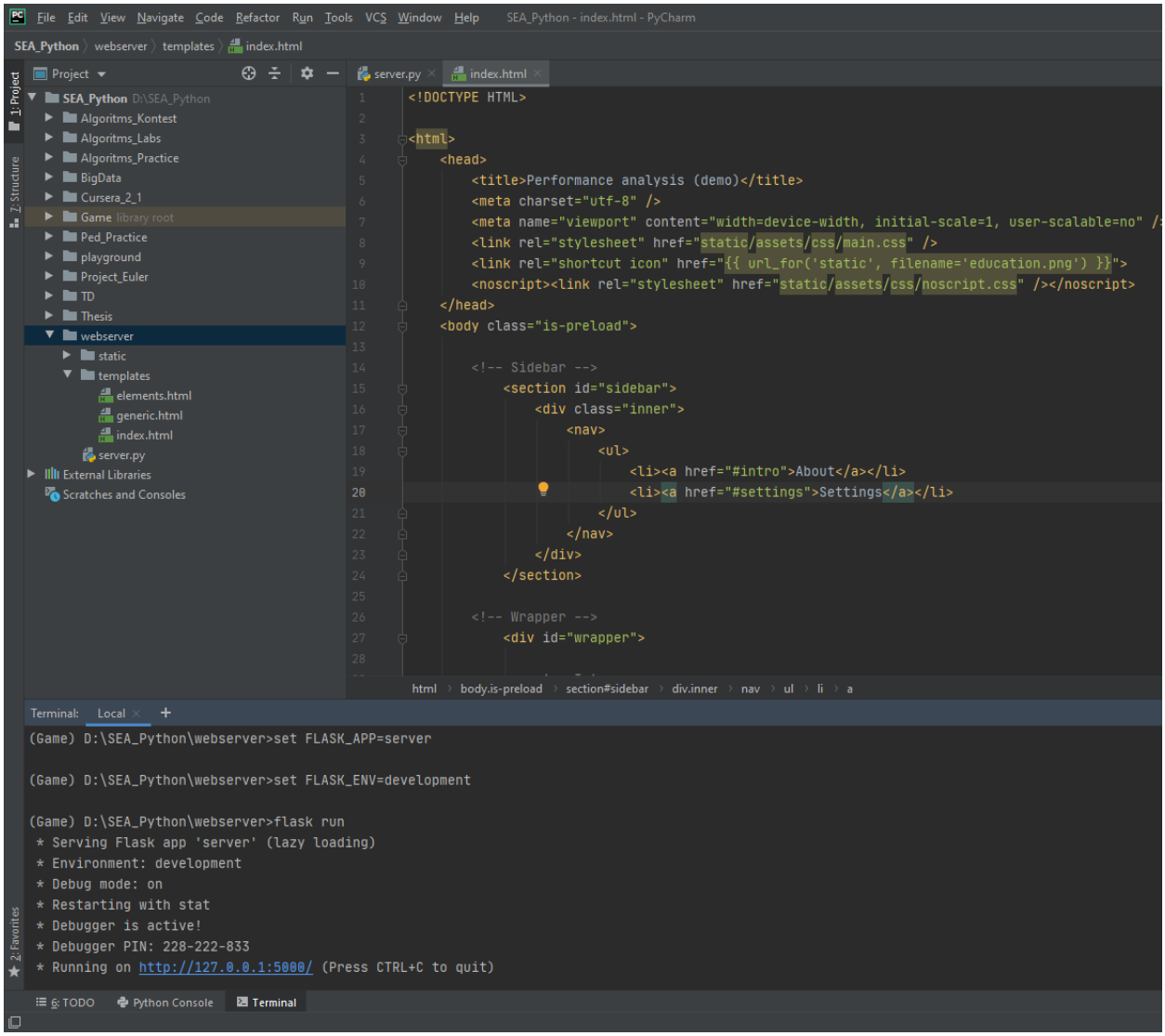


Рисунок №17. Код страницы index.html

При необходимости веб-интерфейс легко расширяется. Например, для ввода данных не об одной паре серверов (сервер приложения и сервер базы данных), а о списке, можно добавить загрузку на страницу списка с данными. В рамках этой диссертации пара серверов будет одна.

MS SQL Dynamic Management Views

MS SQL предоставляет доступ к динамическим административным представлениям (Dynamic Management Views). Это статистический кэш, который хранится до очередной перезагрузки сервера. Его данные можно использовать для диагностики проблем и оптимизации производительности баз данных. Он хранит информацию в специальных таблицах о:

- Причинах задержек выполнения запросов.

- Работе с индексами (отсутствующие, неиспользуемые, требующие больше всех операций ввода/вывода, часто используемые).
- Запросах с высокими издержками на ввод-вывод, с высоким использованием процессора.
- Запросах, выполняющихся чаще всего.
- Запросах, страдающих от блокировок.

Таблицы, которые будут использоваться в этом проекте:

1. sys.dm_exec_query_stats
2. sys.dm_exec_sql_text
3. sys.dm_exec_query_plan

Они предоставляют доступ, соответственно, к статистике выполнения, текстам и планам запросов из кэша. К недостаткам использования DMV относится то, что в кэше сохраняются не все версии планов запросов, и то, что кэш не предназначен для постоянного хранения и может быть очищен.
Вот пример скрипта, написанного на python, который выводит топ самых долгих запросов, основываясь на данных таблиц DMV:

```
import pyodbc
import datetime


def now_time():
    hour = datetime.datetime.now().hour
    minute = datetime.datetime.now().minute
    seconds = datetime.datetime.now().second
    now = f'{hour:02}:{minute:02}:{seconds:02}'
    return now


def delta_hour_min_sec(start, finish):
    delta = finish - start
    total_sec = delta.total_seconds()
    total_hours = int(total_sec // 3600)
    total_min = int((total_sec % 3600) // 60)
    final_sec = int((total_sec % 3600) % 60)
    return f'Needed time: {total_hours} h. {total_min:02} m. {final_sec:02} s.'


def get_top_query_sql():
    sql = """
          SELECT
             topq.statement_text,
             topq.creation_time,
             topq.last_execution_time,
             topq.execution_count,
             topq.avg_worker_time,
             topq.avg_physical_reads,
             topq.avg_logical_reads,
             topq.max_dop,
             topq.plan_handle,
             qp.query_plan
          FROM
          (
             SELECT TOP 20
```

```
                MIN(qstats.statement_text) AS statement_text,
                MIN(qstats.plan_handle) AS plan_handle,
                MIN(qstats.creation_time) AS creation_time,
                MAX(qstats.last_execution_time) AS last_execution_time,
                SUM(qstats.execution_count) AS execution_count,
                SUM(qstats.total_worker_time) / SUM(qstats.execution_count) AS
avg_worker_time,
                SUM(qstats.total_physical_reads) / SUM(qstats.execution_count) AS
avg_physical_reads,
                SUM(qstats.total_logical_reads) / SUM(qstats.execution_count) AS
avg_logical_reads,
                MAX(qstats.max_dop) AS max_dop
            FROM
                (SELECT
                    qs.query_hash,
                    qs.plan_handle,
                    qs.creation_time,
                    qs.last_execution_time,
                    qs.execution_count,
                    qs.total_worker_time,
                    qs.total_physical_reads,
                    qs.total_logical_reads,
                    qs.max_dop,
                    SUBSTRING(st.text, (qs.statement_start_offset/2)+1,
                          ((CASE qs.statement_end_offset
                            WHEN -1 THEN DATALENGTH(st.text)
                           ELSE qs.statement_end_offset
                           END - qs.statement_start_offset)/2) + 1) AS statement_text
                FROM sys.dm_exec_query_stats AS qs
                CROSS APPLY sys.dm_exec_sql_text(qs.plan_handle) AS st

                WHERE st.dbid = DB_ID('OPE_070') ) AS qstats

            GROUP BY qstats.query_hash
            HAVING MIN(qstats.statement_text) LIKE 'SELECT%'
            ORDER BY avg_worker_time DESC

        ) topq

        CROSS APPLY sys.dm_exec_query_plan(topq.plan_handle) qp
        """
    return sql


def connect():
    start = datetime.datetime.now()

    connection_string = "Driver={SQL Server Native Client 11.0};" \
                    "Server=VM-TEST;" "Trusted_Connection=no;" \
                    "UID=sa;" "PWD=***;"
    base_conn = pyodbc.connect(connection_string, autocommit=True)

    cursor = base_conn.cursor()

    cursor.execute(get_top_query_sql())

    databases = cursor.fetchall()
    base_conn.close()

    databases_list = []
    for base in databases:
        databases_list.append(base[0])
```

```
        print(base[1])
        print(base[0])

    #print(databases_list)

    finish = datetime.datetime.now()
    time_delta = delta_hour_min_sec(start, finish)
    print(time_delta)


connect()
```

Пример вывода работы скрипта:
D:/SEA_Python/Thesis/SQL_connector/DMV.py
2021-12-16 13:25:39.160000
SELECT
T1._IDRRef
FROM dbo._Document24631 T1
WHERE (((T1._Fld25280RRef = @P1) AND (T1._Fld25981RRef <> @P2)) OR ((T1._Fld25280RRef = @P3) AND (T1._Fld25981RRef = @P4))) AND (T1._Fld35355RRef = @P5) AND (T1._Marked = 0x00) AND (T1._Fld27381 <= @P6) AND (T1._Fld27381 <> @P7) AND (T1._Fld28763 = 0x00)
GROUP BY T1._IDRRef
2021-12-16 13:25:34.857000
SELECT
T1._Document24631_IDRRef,
T1._Fld24635RRef
FROM dbo._Document24631_VT24633 T1
LEFT OUTER JOIN dbo._Document24631 T2
ON T1._Document24631_IDRRef = T2._IDRRef
WHERE (NOT (((T2._Fld35355RRef = @P1)))) AND (NOT (((T2._Fld35355RRef = @P2)))) AND (NOT (((T2._Fld35355RRef = @P3)))) AND (NOT (((T2._Fld35355RRef = @P4)))) AND (T1._Fld25293RRef = @P5) AND (T2._Marked = 0x00) AND (NOT ((0x0000008B = 0x0000008B AND T1._Fld24635RRef IN
(SELECT
T3._Q_000_F_000RRef AS Q_002_F_000RRef
FROM #tt1 T3 WITH(NOLOCK)))))
2021-12-16 13:25:03.530000
SELECT
T1._Document24631_IDRRef,
T1._Fld24635RRef
FROM dbo._Document24631_VT24633 T1
LEFT OUTER JOIN dbo._Document24631 T2
ON T1._Document24631_IDRRef = T2._IDRRef
WHERE (NOT (((T2._Fld35355RRef = @P1)))) AND (NOT (((T2._Fld35355RRef = @P2)))) AND (NOT (((T2._Fld35355RRef = @P3)))) AND (NOT (((T2._Fld35355RRef = @P4)))) AND (T1._Fld25293RRef = @P5)

AND (T2._Marked = 0x00) AND (NOT ((0x0000008B = 0x0000008B AND
T1._Fld24635RRef IN
(SELECT
T3._Q_001_F_000RRef AS Q_002_F_000RRef
FROM #tt4 T3 WITH(NOLOCK)))))
2021-12-16 13:25:05.893000
SELECT FileName,Creation,Modified,Attributes,DataSize FROM Config
WHERE PartNo = 0 and FileName LIKE @P1
2021-12-16 13:27:03.490000
SELECT
…

Данные по тексту запроса, которые выводит этот скрипт, необходимо обработать для хранения – убрать из него все названия временных таблиц, убрать параметры, структурировать. В результате будет сохраняться некое представление всех запросов и время их выполнения. В дальнейшем из этого топа можно будет собрать топ топов запросов, подлежащих анализу и оптимизации.

MS SQL Extended Events

Благодаря архитектуре MS SQL Extended Events[16] есть возможность собирать трассировку выполнения запросов для устранения неполадок или проблем с производительностью. В данном проекте предусмотрен сбор трассировки по следующим направлениям:

- Анализ ожиданий на блокировках
- Анализ взаимоблокировок
- Анализ «долгих» запросов
- Анализ «тяжелых» запросов по CPU

Вот пример скрипта, написанного на python, который запускает трассировку по всем направлениям:

```
import pyodbc
import datetime


def now_time():
    hour = datetime.datetime.now().hour
    minute = datetime.datetime.now().minute
    seconds = datetime.datetime.now().second
    now = f'{hour:02}:{minute:02}:{seconds:02}'
    return now


def delta_hour_min_sec(start, finish):
    delta = finish - start
```

16 [27] https://docs.microsoft.com/ru-ru/sql/relational-databases/extended-events/extended-events?view=sql-server-2017

```
    total_sec = delta.total_seconds()
    total_hours = int(total_sec // 3600)
    total_min = int((total_sec % 3600) // 60)
    final_sec = int((total_sec % 3600) % 60)
    return f'Needed time: {total_hours} h. {total_min:02} m. {final_sec:02} s.'


def start_session(cursor, name_session):
    sql = """
          ALTER EVENT SESSION [""" + name_session + """] ON SERVER
          STATE = START;
    """
    cursor.execute(sql)


def get_lock_analyze_sql():
    sql = """
        -- Анализ ожиданий
        CREATE EVENT SESSION [LockAnalyze] ON SERVER
        ADD EVENT sqlserver.lock_acquired(

ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.client_pid,sqlserver.
database_id,sqlserver.nt_username,sqlserver.server_principal_name,sqlserver.session_id,sql
server.sql_text,sqlserver.transaction_id,sqlserver.username)),
        ADD EVENT sqlserver.lock_cancel(

ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.client_pid,sqlserver.
database_id,sqlserver.nt_username,sqlserver.server_principal_name,sqlserver.session_id,sql
server.sql_text,sqlserver.transaction_id,sqlserver.username)),
        ADD EVENT sqlserver.lock_timeout(
            WHERE ([duration]>(1) AND [resource_0]<>(0)))
        ADD TARGET package0.event_file(SET
filename=N'D:\SEA_TD_MON\SEA_ExEv\LockAnalyze.xel',max_file_size=(1024),max_rollover_
files=(5))
        WITH (MAX_MEMORY=4096
KB,EVENT_RETENTION_MODE=ALLOW_SINGLE_EVENT_LOSS,MAX_DISPATCH_LATENCY=15
SECONDS,MAX_EVENT_SIZE=0
KB,MEMORY_PARTITION_MODE=NONE,TRACK_CAUSALITY=OFF,STARTUP_STATE=OFF)
        """
    return sql


def get_dead_lock_analyze_sql():
    sql = """
         -- Анализ дедлоков
        CREATE EVENT SESSION [DeadlockAnalyze] ON SERVER
        ADD EVENT sqlserver.lock_deadlock(

ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.client_pid,sqlserver.
database_id,sqlserver.nt_username,sqlserver.server_principal_name,sqlserver.session_id,sql
server.sql_text,sqlserver.transaction_id,sqlserver.username)),
        ADD EVENT sqlserver.lock_deadlock_chain(

ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.client_pid,sqlserver.
database_id,sqlserver.nt_username,sqlserver.server_principal_name,sqlserver.session_id,sql
server.sql_text,sqlserver.transaction_id,sqlserver.username)),
        ADD EVENT sqlserver.xml_deadlock_report(

ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.client_pid,sqlserver.
database_id,sqlserver.nt_username,sqlserver.server_principal_name,sqlserver.session_id,sql
server.sql_text,sqlserver.transaction_id,sqlserver.username))
        ADD TARGET package0.event_file(SET
```

```
filename=N'D:\SEA_TD_MON\SEA_ExEv\DeadlockAnalyze.xel',max_file_size=(10),max_rollover_files=(5))
        WITH (MAX_MEMORY=4096
KB,EVENT_RETENTION_MODE=ALLOW_SINGLE_EVENT_LOSS,MAX_DISPATCH_LATENCY=15
SECONDS,MAX_EVENT_SIZE=0
KB,MEMORY_PARTITION_MODE=NONE,TRACK_CAUSALITY=OFF,STARTUP_STATE=OFF)
        """
    return sql


def get_heavy_query_by_cpu():
    sql = """
         -- Анализ запросов по нагрузке процессора
        CREATE EVENT SESSION [HeavyQueryByCPU] ON SERVER
        ADD EVENT sqlserver.rpc_completed(

ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.client_pid,sqlserver.database_id,sqlserver.nt_username,sqlserver.server_principal_name,sqlserver.session_id,sqlserver.sql_text,sqlserver.transaction_id,sqlserver.username)
            WHERE ([duration]>(50000))),
        ADD EVENT sqlserver.sql_batch_completed(

ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.client_pid,sqlserver.database_id,sqlserver.nt_username,sqlserver.server_principal_name,sqlserver.session_id,sqlserver.sql_text,sqlserver.transaction_id,sqlserver.username)
            WHERE ([duration]>(50000)))
        ADD TARGET package0.event_file(SET
filename=N'D:\SEA_TD_MON\SEA_ExEv\HeavyQueryByCPU.xel',max_file_size=(1024),max_rollover_files=(5))
        WITH (MAX_MEMORY=4096
KB,EVENT_RETENTION_MODE=ALLOW_SINGLE_EVENT_LOSS,MAX_DISPATCH_LATENCY=15
SECONDS,MAX_EVENT_SIZE=0
KB,MEMORY_PARTITION_MODE=NONE,TRACK_CAUSALITY=OFF,STARTUP_STATE=OFF)
        """
    return sql


def connect():
    start = datetime.datetime.now()
    connection_string = "Driver={SQL Server Native Client 11.0};" \
                        "Server=VM-TEST;" "Trusted_Connection=no;" \
                        "UID=sa;" "PWD=***;"
    base_conn = pyodbc.connect(connection_string, autocommit=True)

    cursor = base_conn.cursor()

    cursor.execute('SELECT name FROM  sys.server_event_sessions')

    ex_events = cursor.fetchall()

    ex_events_list = []
    for ex_event in ex_events:
        ex_events_list.append(ex_event[0])

    if "DeadlockAnalyze" in ex_events_list:
        start_session(cursor, "DeadlockAnalyze")
    else:
        cursor.execute(get_dead_lock_analyze_sql())

    if "HeavyQueryByCPU" in ex_events_list:
        start_session(cursor, "HeavyQueryByCPU")
    else:
```

```
        cursor.execute(get_heavy_query_by_cpu())

    base_conn.close()
    finish = datetime.datetime.now()
    time_delta = delta_hour_min_sec(start, finish)
    print(time_delta)
connect()
```

Текст таких подозрительных запросов, полученный из лог-файлов трассировки, необходимо (аналогично тексту запросов DMV) обработать для хранения – убрать из него все названия временных таблиц, убрать параметры, структурировать. В результате будет сохраняться некое представление всех запросов и время их выполнения. В дальнейшем эту информацию будет анализировать ядро проекта и предоставлять визуализацию итога через дашборды Grafana.

Счетчики производительности

Довольно мощным удобным и от того одним из самых необходимых мощных инструментов для обнаружения проблем с производительностью в операционной системе Windows являются встроенные счетчики производительности. Встроенным инструментом для управления счетчиками производительности служит оснастка «Монитор производительности (Performance Monitor)». Этот инструмент может работать в двух режимах – монитор и сбор данных. На рисунке №18 показан скриншот собранных за период данных.

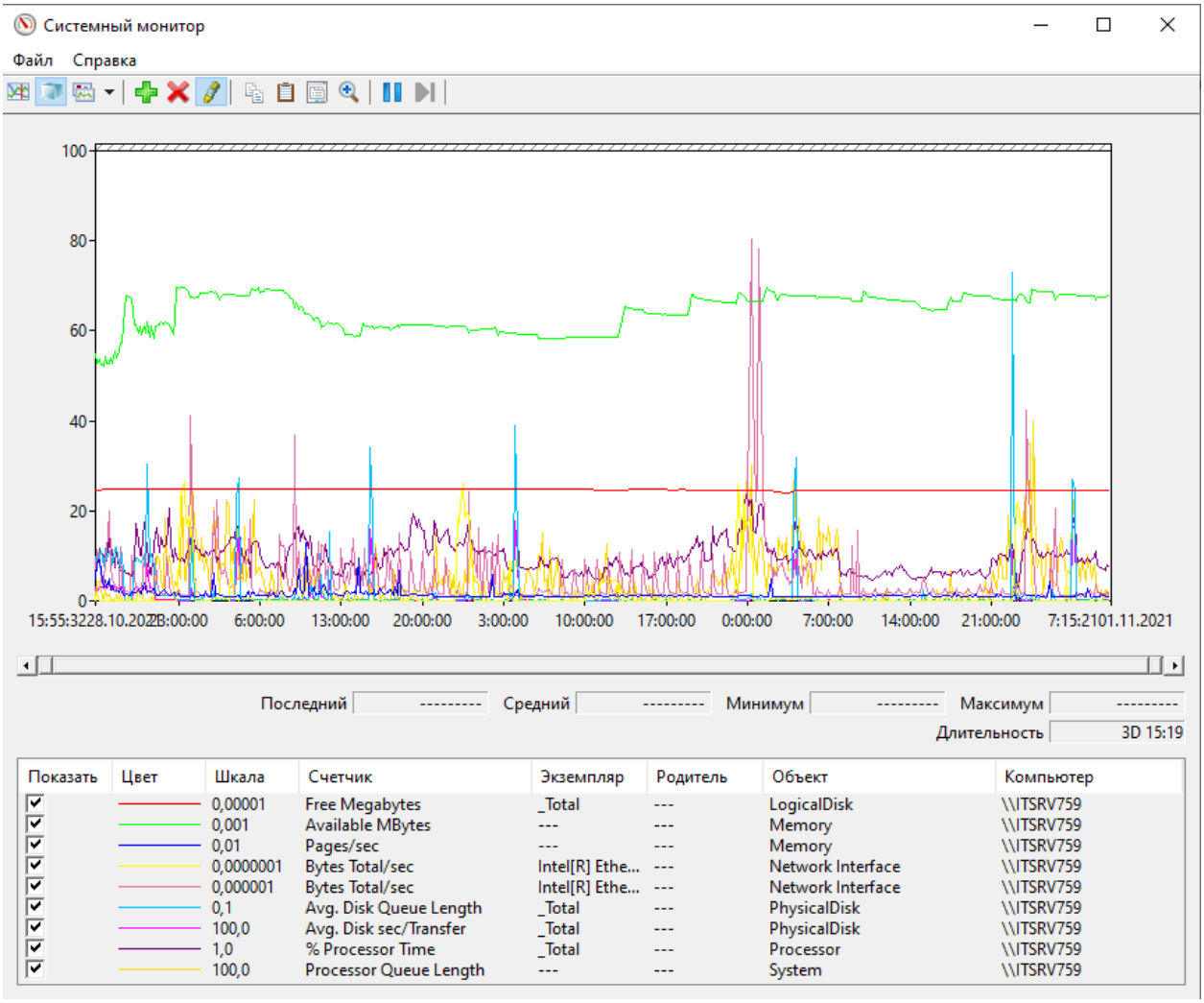


Рисунок №18. Системный монитор

Кроме ручной настройки и запуска сбора счетчиков производительности разработчиками этого инструмента предусмотрена программная возможность создания, запуска и сохранения данных по счетчикам. Эту возможность будет использовать данный проект для программного сбора информации с последующим выводом в общий график.

Будут использованы следующие счетчики производительности:

1. Длинна очереди процессора
2. Процент загруженности процессора
3. Среднее время обращения к диску
4. Свободно мегабайт на диске
5. Средняя длина очереди диска
6. Доступно МБ оперативной памяти
7. Обмен страниц оперативной памяти

При написании скрипта, запускающего сбор данных по этим счетчикам, был использован в качестве примера код из одного из сайтов[17] на просторах интернета. Ниже приведена часть скрипта с описанием класса «PerfMonitor»

17 https://sudonull.com/post/118722-Experience-using-logman-to-collect-application-performance-metrics-on-Windows

```
import os
import subprocess
import glob
import time


COUNTER_Processor_ProcessorTime = '"\Processor(_Total)\% Processor Time"'
COUNTER_System_ProcessorQueueLength = '"\System\Processor Queue Length"'
COUNTER_Memory_AvailableMBytes = '"\Memory\Available MBytes"'
COUNTER_Memory_Pages = '"\Memory\Pages/sec"'
COUNTER_LogicalDisk_FreeMegabytes = '"\LogicalDisk(_Total)\Free Megabytes"'
COUNTER_PhysicalDisk_AvgDisk = '"\PhysicalDisk(_Total)\Avg. Disk sec/Transfer"'
COUNTER_PhysicalDisk_AvgDiskQueueLength = '"\PhysicalDisk(_Total)\Avg. Disk Queue
Length"'


class PerfMonitor(object):
    def __init__(self, results_dir, max_mem=1, interval=5):
        self._counters = {'cpu.time': COUNTER_Processor_ProcessorTime,
                          'cpu.queue': COUNTER_System_ProcessorQueueLength,
                          'mem.available': COUNTER_Memory_AvailableMBytes,
                          'mem.pages': COUNTER_Memory_Pages,
                          'disk.free': COUNTER_LogicalDisk_FreeMegabytes,
                          'disk.avg': COUNTER_PhysicalDisk_AvgDisk,
                          'disk.queue': COUNTER_PhysicalDisk_AvgDiskQueueLength}
        self._results_dir = os.path.normpath(results_dir)
        self._raw_dir = os.path.join(self._results_dir, 'raw')
        self._final_dir = os.path.join(self._results_dir, 'csv')
        self._interval = interval
        self._max_mem = max_mem
        self._collectors = []
        if not os.path.exists(self._results_dir):
            os.makedirs(self._results_dir)
            os.makedirs(self._raw_dir)
            os.makedirs(self._final_dir)

    def init_all(self):
        for lang in self._counters:
            collector = 'sea-{collector}'.format(collector=lang)
            output = os.path.join(self._raw_dir, collector + '.csv')
            self.__create(collector, self._counters[lang], output)
            self._collectors.append(collector)

    def clean_all(self):
        for collector in self._collectors:
            csv = os.path.join(self._raw_dir, '{}.csv'.format(collector))
            if os.path.isfile(csv):
                os.remove(csv)

    def start_all(self):
        for collector in self._collectors:
            self.__start(collector)

    def stop_all(self):
        for collector in self._collectors:
            self.__stop(collector)
            self.__delete(collector)

    def make_csv(self):
        for lang in self._counters:
            final_output = os.path.join(self._final_dir, '{}.csv'.format(lang))
            self.__format_csv(final_output, lang)
```

```
    def __format_csv(self, output, counter):
        with open(output, 'a') as outfile:
            for csv in glob.glob(os.path.join(self._raw_dir, '*{}*'.format(counter))):
                with open(csv, 'r') as file:
                    outfile.writelines(file.readlines()[1:])

    def __create(self, name, counter, output):
        cmd = 'logman create counter "{name}" -f csv -si {interval} ' \
              '--v -o "{output}" -c {counter}'.format(name=name,
                                                     interval=str(self._interval),
                                                     output=output,
                                                     counter=counter)
        print(cmd)
        subprocess.check_call(cmd)

    def __start(self, collector):
        cmd = 'logman start {}'.format(collector)
        subprocess.check_call(cmd)

    def __stop(self, collector):
        cmd = 'logman stop {}'.format(collector)
        subprocess.check_call(cmd)

    def __delete(self, collector):
        cmd = 'logman delete {}'.format(collector)
        subprocess.check_call(cmd)


def start_monitoring_logman_app(name_comp):
    permon = PerfMonitor('D:/SEA_TD_MON/SEA_MON/')
    permon.init_all()
    permon.start_all()
    time.sleep(60)
    permon.stop_all()
    permon.make_csv()
```

На рисунке №19 можно увидеть результат работы этого скрипта

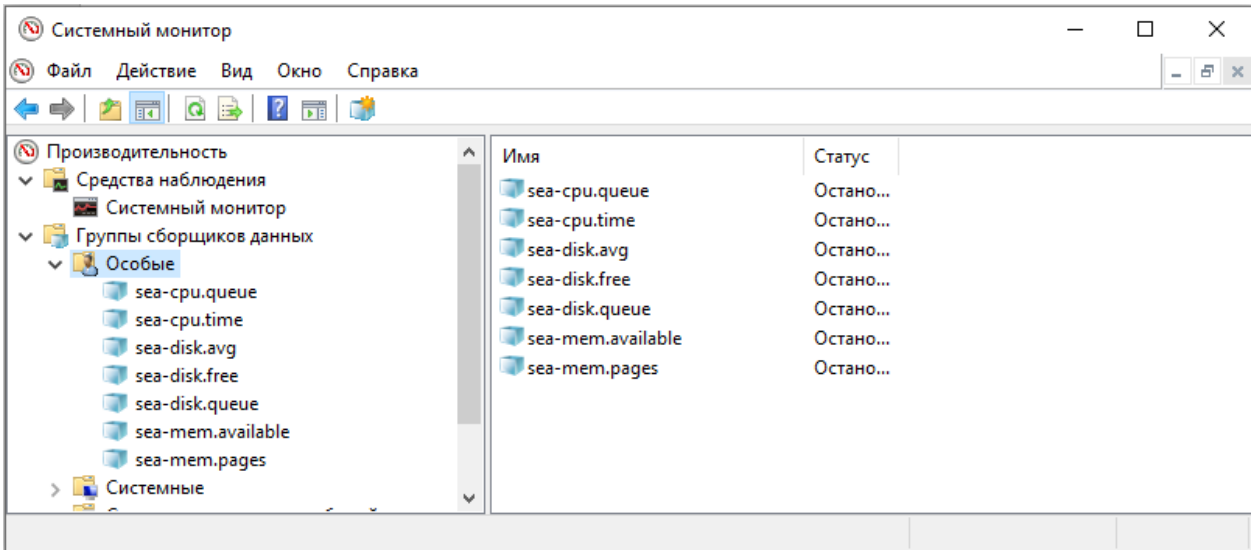


Рисунок №19. Результат работы скрипта

Информация после отработки скрипта, который запускает счетчики и сохраняет результат их работы, будет в файлах csv примерно вот в таком формате:

```
03/14/2022 13:04:45.080,"0.00030913437499999999077"
03/14/2022 13:04:50.088,"0.00030611360544217686326"
03/14/2022 13:04:55.082,"0.00017767971014492754027"
03/14/2022 13:05:00.076,"9.939185185185184527 5e-005"
03/14/2022 13:05:05.087,"0.00013954637681159421064"
03/14/2022 13:05:10.081,"9.792499999999995013e-005"
03/14/2022 13:05:15.074,"6.5426086956521742615e-005"
03/14/2022 13:05:20.082,"0.00024774635761589400632"
03/14/2022 13:05:25.076,"7.589811320754717509 9e-005"
03/14/2022 13:05:30.085,"7.5954838709677417632e-005"
```

Технологический журнал

Клиентское приложение: «1С:Предприятие 8.3», производительность работы которого является предметом исследования данной магистерской диссертации, имеет свой инструмент логирования – «Технологический журнал». Он представляет собой совокупность текстовых файлов, хранящихся в указанном каталоге и зависящих от настроек его конфигурационного файла.

События технологического журнала, которые будут анализироваться:

1. CALL – вызов серверного контекста. Позволяет получить данные о длительности и количестве используемой оперативной памяти серверного вызова. Будет использоваться для анализа производительности сервера 1С.
2. DBMSSQL – работа запроса в MS SQL. Позволяет получить данные о длительности выполнения запроса. Будет использоваться для анализа производительности сервера БД.
3. EXCP – ошибка работы приложения. Будет использоваться для анализа количества ошибок

На рисунке №20 пример конфигурационного файла для сбора данных по этим событиям.

```
<?xml version="1.0"?>
- <config xmlns="http://v8.1c.ru/v8/tech-log">
  - <log location="C:\SEA_MON\TJ_LOG" history="8">
    - <event>
        <eq value="CALL" property="name"/>
      </event>
    - <event>
        <eq value="DBMSSQL" property="name"/>
      </event>
    - <event>
        <eq value="EXCP" property="name"/>
      </event>
      <property name="all"/>
    </log>
</config>
```

Рисунок №20. Настройки ТЖ

Для запуска технологического журнала нужно конфигурационный файл поместить в соответствующую папку. Пример скрипта, который это делает:

```
def start_monitoring_tj(name_comp):
    xml_str = '<?xml version="1.0"?>' + '\n' + \
            '<config xmlns="http://v8.1c.ru/v8/tech-log">' + '\n' + \
            '<log history="8" location="C:\\SEA_MON\\TJ_CALL">' + '\n' + \
            '<event>' + '\n' + \
            '<eq property="name" value="CALL"/>' + '\n' + \
            '</event>' + '\n' + \
            '<event>' + '\n' + \
            '<eq property="name" value="DBMSSQL"/>' + '\n' + \
            '</event>' + '\n' + \
            '<event>' + '\n' + \
            '<eq property="name" value="EXCP"/>' + '\n' + \
            '</event>' + '\n' + \
            '<property name="all"/>' + '\n' + \
            '</log>' + '\n' + \
            '</config>'

    f = open('C:/Program Files/1cv8/conf/logcfg.xml', 'w')
    f.write(xml_str)
    f.close()
```

На рисунке №21 пример получаемых данных.

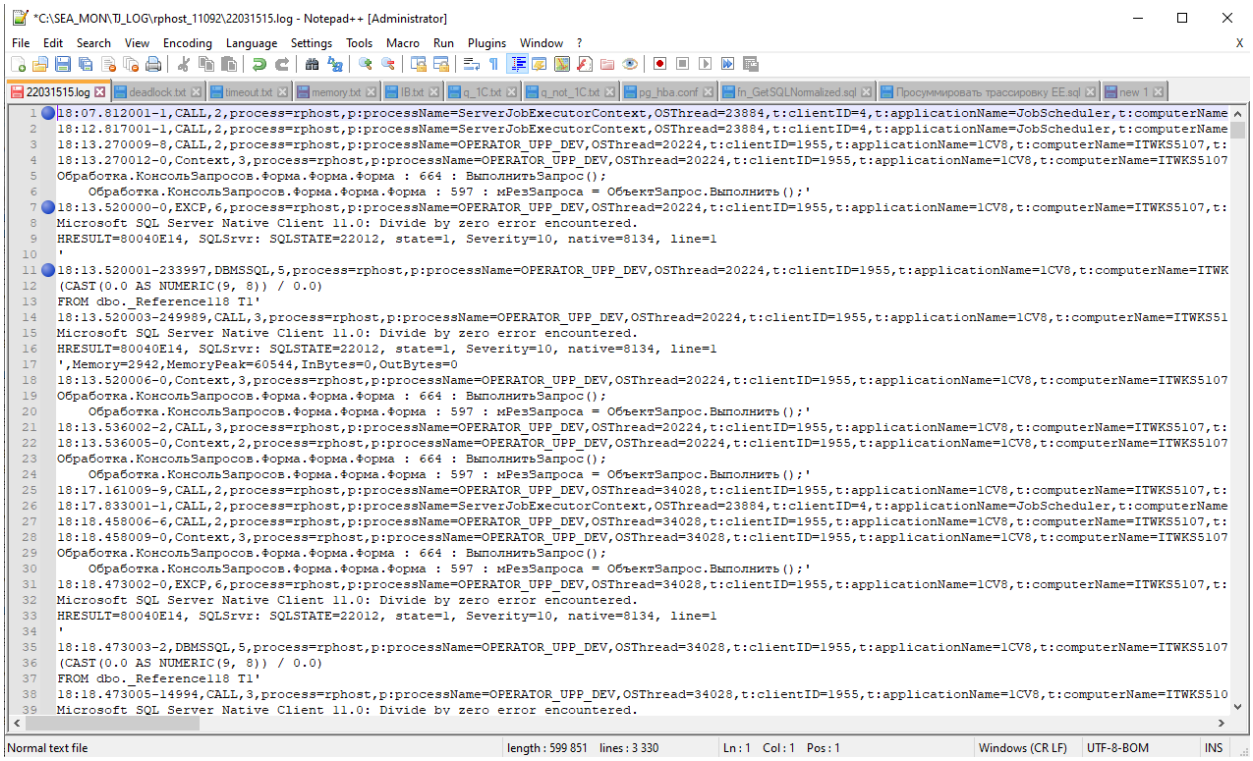


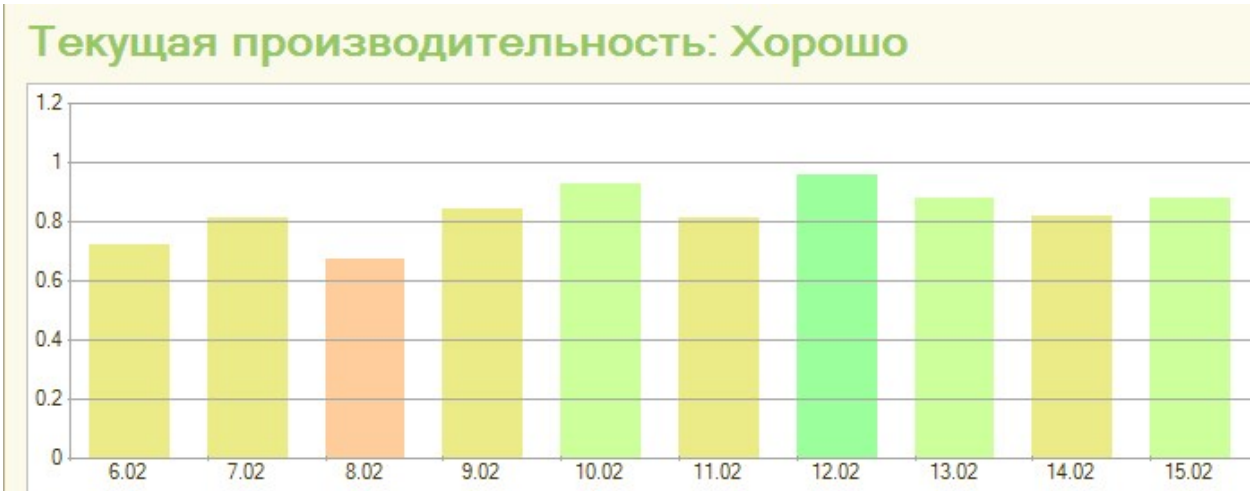
Рисунок №21. Собранные логи ТЖ

Возможности технологического журнала очень большие. В данной диссертации рассматривается лишь малая их часть. Скорее всего, в последствии во время активного использования полученного программного продукта на продуктивном контуре в анализ добавится сбор других событий. Например, для нахождения циклических ссылок (SCRIPTCIRCREFS), ожиданий на избыточных блокировках (TTIMEOUT), взаимоблокировок (TDEADLOCK) и утечек памяти (LEAKS).

Apdex

В клиентское приложение «1С:Предприятие 8.3» внедрена система Apdex (Application Performance Index) – индекс производительности приложений. Если не вдаваться в подробности, то эта система подразумевает сбор данных о планируемом и фактическом времени выполнения ключевых операций. То есть, пользователи приложения определяют список ключевых операций и время, за которое каждая из операций должна выполняться. Например, кассир говорит, что документ «Приходный кассовый ордер» должен сохраняться в идеале за три секунду, или за четыре – это чуть хуже, или за пять – это плохо, но приемлемо, или больше пяти секунд, что уже неприемлемо. Из этих плановых данных и полученных во время работы приложения фактических данных высчитывается индекс. По его значению можно понимать степень приемлемости работы приложения в целом и по каждой операции в отдельности.

На рисунке №22, взятом с сайта gilev.ru[18] показан пример отчета производительности системы по показателям Apdex



Расшифровка производительности по операциям

Показатель	6.02	7.02	8.02	9.02	10.02	11.02	12.02	13.02	14.02	15.02
ЗаписьПроектаИзФормы	плохо	неуд.	неуд.	плохо	отл.	неуд.	плохо	плохо		плохо
ОктрытиеСпискаПроектов	хор.	плохо	плохо		хор.		отл.	плохо		отл.
ОктрытиеЭлементаСправочникаКонтрагенты	отл.	отл.	отл.		отл.					отл.
Поступление товаров услуг	отл.	отл.	отл.	отл.	отл.	отл.	отл.	отл.	отл.	отл.
Расчет себестоимости товаров	отл.	отл.	отл.	отл.	отл.	отл.	отл.	отл.	отл.	отл.
ЗаписьЗаданияИзФормы	неуд.	плохо	неуд.	неуд.	неуд.	плохо		плохо	неуд.	плохо
Общий % за день	**плохс**	**плохс**	**плохс**	**плохс**	**хор.**	**плохс**	**отл.**	**хор.**	**плохс**	**хор.**

Рисунок №22. Пример отчета по Apdex

Пример кода, который выгружает данные по Apdex в файлы csv для последующей их обработки и выгрузки в Prometheus:

```
Процедура ПолучитьДанныеИВыгрузитьИхВCSV()

	Запрос = Новый Запрос;
	Запрос.Текст =
	"ВЫБРАТЬ
	|	ЗамерыВремени.КлючеваяОперация КАК КлючеваяОперация,
	|	ЗамерыВремени.КлючеваяОперация.ЦелевоеВремя КАК ЦелевоеВремя,
	|	ЗамерыВремени.КлючеваяОперация.Приоритет КАК Приоритет,
	|	ЗамерыВремени.ДатаЗаписи КАК ДатаЗаписи,
	|	ЗамерыВремени.ВремяВыполнения КАК ВремяВыполнения,
	|	ВЫБОР
	|		КОГДА ЗамерыВремени.ВремяВыполнения <=
ЗамерыВремени.КлючеваяОперация.ЦелевоеВремя
	|			ТОГДА 1
	|		ИНАЧЕ 0
	|	КОНЕЦ КАК КоличествоОперацийОтлично,
	|	ВЫБОР
	|		КОГДА ЗамерыВремени.ВремяВыполнения >
ЗамерыВремени.КлючеваяОперация.ЦелевоеВремя
	|				И ЗамерыВремени.ВремяВыполнения <=
ЗамерыВремени.КлючеваяОперация.ЦелевоеВремя * 4
```

18 http://www.gilev.ru/apdex-teoriya/

```
        |                       ТОГДА 1
        |                   ИНАЧЕ 0
        |               КОНЕЦ КАК КоличествоОперацийХорошо,
        |               0 КАК APDEX,
        |               1 КАК КоличествоОпераций,
        |               ЗамерыВремени.Пользователь КАК Пользователь,
        |               ЗамерыВремени.НомерСеанса КАК НомерСеанса,
        |               НАЧАЛОПЕРИОДА(ЗамерыВремени.ДатаЗаписи, ЧАС) КАК
ДатаЗаписиЧас,
        |               НАЧАЛОПЕРИОДА(ЗамерыВремени.ДатаЗаписи, ДЕНЬ) КАК
ДатаЗаписиДень,
        |               НАЧАЛОПЕРИОДА(ЗамерыВремени.ДатаЗаписи, МЕСЯЦ) КАК
ДатаЗаписиМесяц,
        |               НАЧАЛОПЕРИОДА(ЗамерыВремени.ДатаЗаписи, ГОД) КАК
ДатаЗаписиГод,
        |               НАЧАЛОПЕРИОДА(ЗамерыВремени.ДатаЗаписи, МИНУТА) КАК
ДатаЗаписиМинута,
        |               ЗамерыВремени.ДатаЗаписи КАК ДатаЗаписиСекунда,
        |               ЛОЖЬ КАК Технологическая
        |ПОМЕСТИТЬ ИсходныеДанные
        |ИЗ
        |               РегистрСведений.ЗамерыВремени КАК ЗамерыВремени
        |ГДЕ
        |               ЗамерыВремени.ДатаЗаписи МЕЖДУ &НачалоПериода И &КонецПериода
        |;
        |
        |////////////////////////////////////////////////////////////////////////////
        |ВЫБРАТЬ
        |               ИсходныеДанные.КлючеваяОперация КАК КлючеваяОперация,
        |               ИсходныеДанные.ЦелевоеВремя КАК ЦелевоеВремя,
        |               ИсходныеДанные.Приоритет КАК Приоритет,
        |               ИсходныеДанные.ДатаЗаписи КАК ДатаЗаписи,
        |               ИсходныеДанные.ВремяВыполнения КАК ВремяВыполнения,
        |               ИсходныеДанные.КоличествоОперацийОтлично КАК
КоличествоОперацийОтлично,
        |               ИсходныеДанные.КоличествоОперацийХорошо КАК
КоличествоОперацийХорошо,
        |               ИсходныеДанные.APDEX КАК APDEX,
        |               ИсходныеДанные.КоличествоОпераций КАК КоличествоОпераций,
        |               ИсходныеДанные.Пользователь КАК Пользователь,
        |               ИсходныеДанные.НомерСеанса КАК НомерСеанса,
        |

ЕСТЬNULL(ОбщиеДанныеКлючевыхОпераций.МаксимальноеКоличествоОперацийЗаПер
иод, 0) КАК МаксимальноеКоличествоОпераций,
        |               ВЫБОР
        |                       КОГДА
ЕСТЬNULL(ОбщиеДанныеКлючевыхОпераций.МинимальныйПриоритет, 0) = 0
        |                               ТОГДА 1
        |                       ИНАЧЕ
ЕСТЬNULL(ОбщиеДанныеКлючевыхОпераций.МинимальныйПриоритет, 0)
        |               КОНЕЦ КАК МинимальныйПриоритет,
```

```
        |        ИсходныеДанные.ДатаЗаписиЧас КАК ДатаЗаписиЧас,
        |        ИсходныеДанные.ДатаЗаписиДень КАК ДатаЗаписиДень,
        |        ИсходныеДанные.ДатаЗаписиМесяц КАК ДатаЗаписиМесяц,
        |        ИсходныеДанные.ДатаЗаписиГод КАК ДатаЗаписиГод,
        |        ИсходныеДанные.ДатаЗаписиМинута КАК ДатаЗаписиМинута,
        |        ИсходныеДанные.ДатаЗаписиСекунда КАК ДатаЗаписиСекунда,
        |        ИсходныеДанные.Технологическая КАК Технологическая
        |ИЗ
        |        ИсходныеДанные КАК ИсходныеДанные
        |                ЛЕВОЕ СОЕДИНЕНИЕ (ВЫБРАТЬ
        |
        МАКСИМУМ(ДанныеКлючевыхОпераций ДляГруппировки.КоличествоОпераций)
КАК МаксимальноеКоличествоОпераций ЗаПериод,
        |
        МАКСИМУМ(ДанныеКлючевыхОпераций ДляГруппировки.Приоритет) КАК
МинимальныйПриоритет
        |                ИЗ
        |                        (ВЫБРАТЬ
        |                                ИсходныеДанные.КлючеваяОперация КАК
КлючеваяОперация,
        |                                СУММА(ИсходныеДанные.КоличествоОпераций) КАК
КоличествоОпераций,
        |                                МИНИМУМ(ИсходныеДанные.Приоритет) КАК
Приоритет
        |                        ИЗ
        |                                ИсходныеДанные КАК ИсходныеДанные
        |
        |                        СГРУППИРОВАТЬ ПО
        |                                ИсходныеДанные.КлючеваяОперация) КАК
ДанныеКлючевыхОпераций ДляГруппировки) КАК ОбщиеДанныеКлючевыхОпераций
        |                ПО (ИСТИНА)";

        РезультатЗапроса = Запрос.Выполнить().Выгрузить();
        ТЗ = РезультатЗапроса;

        ВыгрузитьВCSV(ТЗ)

КонецПроцедуры

Процедура ВыгрузитьВCSV(ТЗ);

  Разделитель = ";";

  ФайлCSV = "D:\SEA_TD_MON\SEA_APDEX\apdex_" + Формат(ТекущаяДата(),
"ДФ=yyyyMMdd") + ".csv";

  ТекстCSV = "";

  Для Каждого СТ Из ТЗ Цикл
    Если ТекстCSV = "" Тогда

      СтрокаКол = "";
```

```
        КолонкиТЗ = ТЗ.Колонки;

                Для Каждого Колонка Из КолонкиТЗ Цикл
            СтрокаКол = "" + СтрокаКол + Колонка.Имя + Разделитель ;
                КонецЦикла;

        ТекстCSV = СтрокаКол + Символы.ПС;

    КонецЕсли;

    ТекстCSV = ТекстCSV + СТ.Операция + Разделитель + СТ.Апдекс + Символы.ПС;

КонецЦикла;

Запись = Новый ЗаписьТекста(ФайлCSV, КодировкаТекста.ANSI);
Запись.ЗаписатьСтроку(ТекстCSV);
Запись.Закрыть();

КонецПроцедуры
```

Мониторинг производительности

После сбора данных продуктивного контура:

1. MS SQL Extended Events
2. MS SQL Dynamic Management Views
3. Счетчики производительности
4. Технологический журнал
5. Apdex

Необходимо преобразовать их в метрики и передать визуализатору Grafana для отображения в настроенных под нужды пользователя системы дашборды. В рамках этой диссертации Grafana будет получать метрики из системы мониторинга Prometheus. Которая в свою очередь будет тянуть (pull) данные из Pushgateway – сервиса, куда можно скидывать (push) метрики. А в этот сервис метрики будут попадать при помощи скрипта на python, который будет парсить собранные данные продуктивного контура компонвать из них метрики и отправлять по адресу, где установлен Pushgateway (http://vm-test-sea:9091/).
На рисунке №23 показаны настройки конфигурационного файла Prometheus.

```
# my global config
global:
  scrape_interval: 5s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 5s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "pushgateway"
    honor_labels: true
   # metrics_path defaults to '/metrics'
   # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9091"]
```

Рисунок №23. Настройки Prometheus

Настройки Grafana на источник данных Prometheus показаны на рисунке №24.

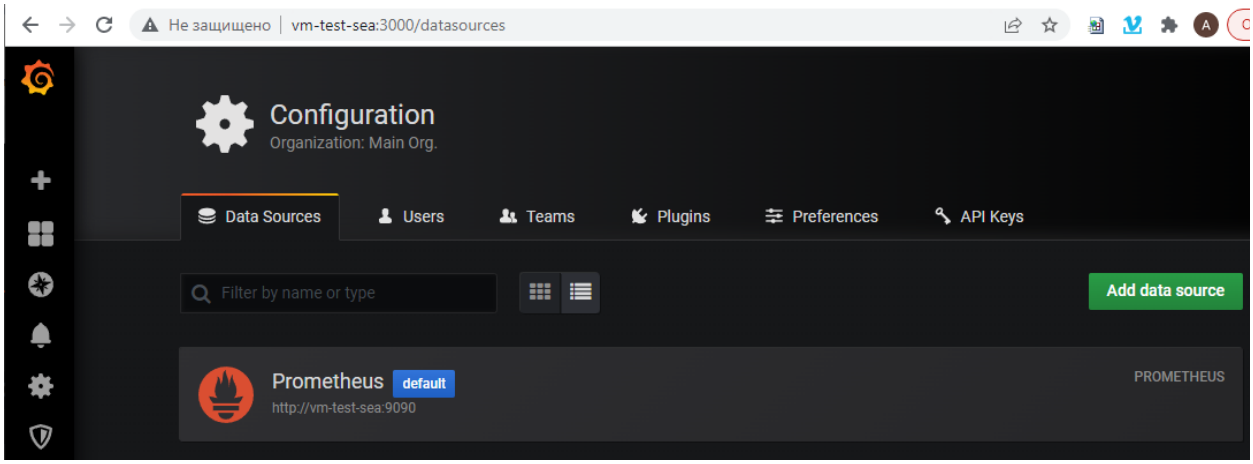


Рисунок №24. Настройки источника данных

Скрипт, который преобразует собранные данные продуктивного контура в метрики:

```
from prometheus_client import CollectorRegistry, Gauge, push_to_gateway
import csv
import time
import pyodbc
import glob
import os
import re

def push():
    connection_string = "Driver={SQL Server Native Client 11.0};" \
                        "Server= VM-TEST;" "Trusted_Connection=no;" \
```

```
                "UID=sa;" "PWD=***;"
    db_conn = pyodbc.connect(connection_string, autocommit=True)

    cursor = db_conn.cursor()

    cursor.execute("""  select
                    xel.sec_5 as sec_5,
                    SUM(xel.duration) as duration,
                    SUM(xel.cpu_time) as cpu_time,
                    SUM(xel.physical_reads) as physical_reads,
                    SUM(xel.logical_reads) as logical_reads,
                    SUM(xel.writes) as writes,
                    SUM(xel.row_count) as row_count

                    FROM (
                    select
                        ROUND((DATEDIFF(SECOND, n.value('(@timestamp)[1]', 'datetime2'),
'01.01.2023'))/5, 0) * 5 AS [sec_5],
                        n.value('(data[@name="duration"]/value)[1]', 'int') as duration,
                        n.value('(data[@name="cpu_time"]/value)[1]', 'int') as cpu_time,
                        n.value('(data[@name="physical_reads"]/value)[1]', 'int') as
physical_reads,
                        n.value('(data[@name="logical_reads"]/value)[1]', 'int') as logical_reads,
                        n.value('(data[@name="writes"]/value)[1]', 'int') as writes,
                        n.value('(data[@name="row_count"]/value)[1]', 'int') as row_count
                    from (select cast(event_data as XML) as event_data
                    from
sys.fn_xe_file_target_read_file('C:\SEA_MON\SEA_ExEv\SEA_Query_Text_*.xel', null, null,
null)) ed
                    cross apply ed.event_data.nodes('event') as q(n)) as xel
                group by sec_5
                order by sec_5 desc""")

    ex_events = cursor.fetchall()
    db_conn.close()

    list_call = []
    list_excp = []
    list_dbmssql = []

    for directory in glob.glob(r'C:\SEA_MON\TJ_LOG\rphos*'):
        files = os.listdir(directory)
        for file_path in files:
            with open(directory + '\\' + file_path, encoding='utf-8') as file:
                for line in file:
                    if re.search(r'\d\d:\d\d\.\d.*?-\d.*,CALL', line):
                        list_split = line.split(',')
                        event = list_split[1]
                        time_list = list_split[0].split('-')
                        try:
                            time_in_second = round((int(time_list[0][0:2]) * 60 + int(time_list[0]
[3:5])) / 5) * 5
                        except:
                            continue
                        duration = int(time_list[1])
                        if duration > 1:
                            list_call.append([str(time_in_second), duration])
                    elif re.search(r'\d\d:\d\d\.\d.*?-\d.*,EXCP', line):
                        list_split = line.split(',')
                        event = list_split[1]
                        time_list = list_split[0].split('-')
                        try:
```

```
                    time_in_second = round((int(time_list[0][0:2]) * 60 + int(time_list[0]
[3:5])) / 5) * 5
                except:
                    continue
                duration = 1
                list_excp.append([str(time_in_second), duration])
            elif re.search(r'\d\d:\d\d\.\d.*?-\d.*,DBMSSQL', line):
                list_split = line.split(',')
                event = list_split[1]
                time_list = list_split[0].split('-')
                try:
                    time_in_second = round((int(time_list[0][0:2]) * 60 + int(time_list[0]
[3:5])) / 5) * 5
                except:
                    continue
                duration = int(time_list[1])
                list_dbmssql.append([str(time_in_second), duration])

    m = dict()
    for x in list_call:
        m.setdefault(x[0], 0)
        m[x[0]] += x[1]
    total_list_call = sorted(list(x) for x in m.items())
    total_list_call.sort()

    m = dict()
    for x in list_excp:
        m.setdefault(x[0], 0)
        m[x[0]] += x[1]
    total_list_excp = sorted(list(x) for x in m.items())
    total_list_excp.sort()

    m = dict()
    for x in list_dbmssql:
        m.setdefault(x[0], 0)
        m[x[0]] += x[1]
    total_list_dbmssql = sorted(list(x) for x in m.items())
    total_list_dbmssql.sort()


    cpu_queue_csv = open('C://SEA_MON//SEA_Logman//csv//cpu.queue.csv', 'r')
    cpu_time_csv = open('C://SEA_MON//SEA_Logman//csv//cpu.time.csv', 'r')
    disk_avg_csv = open('C://SEA_MON//SEA_Logman//csv//disk.avg.csv', 'r')
    disk_free_csv = open('C://SEA_MON//SEA_Logman//csv//disk.free.csv', 'r')
    disk_queue_csv = open('C://SEA_MON//SEA_Logman//csv//disk.queue.csv', 'r')
    mem_available_csv = open('C://SEA_MON//SEA_Logman//csv//mem.available.csv', 'r')
    mem_pages_csv = open('C://SEA_MON//SEA_Logman//csv//mem.pages.csv', 'r')

    reader_cpu_queue_csv = csv.DictReader(cpu_queue_csv)
    reader_cpu_time_csv = csv.DictReader(cpu_time_csv)
    reader_disk_avg_csv = csv.DictReader(disk_avg_csv)
    reader_disk_free_csv = csv.DictReader(disk_free_csv)
    reader_disk_queue_csv = csv.DictReader(disk_queue_csv)
    reader_mem_available_csv = csv.DictReader(mem_available_csv)
    reader_mem_pages_csv = csv.DictReader(mem_pages_csv)

    arr_cpu_queue_csv = []
    arr_cpu_time_csv = []
    arr_disk_avg_csv = []
    arr_disk_free_csv = []
    arr_disk_queue_csv = []
    arr_mem_available_csv = []
```

```
    arr_mem_pages_csv = []

    for each in reader_cpu_queue_csv:
        arr_cpu_queue_csv.append(float(list(each.values())[1]))
    for each in reader_cpu_time_csv:
        arr_cpu_time_csv.append(float(list(each.values())[1]))
    for each in reader_disk_avg_csv:
        arr_disk_avg_csv.append(float(list(each.values())[1])*100000)
    for each in reader_disk_free_csv:
        arr_disk_free_csv.append(float(list(each.values())[1])/100000)
    for each in reader_disk_queue_csv:
        arr_disk_queue_csv.append(float(list(each.values())[1])*1000)
    for each in reader_mem_available_csv:
        arr_mem_available_csv.append(float(list(each.values())[1])/1000)
    for each in reader_mem_pages_csv:
        arr_mem_pages_csv.append(float(list(each.values())[1]))

    len_arr = min(len(arr_cpu_queue_csv), len(arr_cpu_time_csv), len(arr_disk_avg_csv),
len(arr_disk_free_csv),
                  len(arr_disk_queue_csv), len(arr_mem_available_csv), len(arr_mem_pages_csv),
len(ex_events))

    for i in range(len_arr):
        print('------------------------')
        print(arr_cpu_queue_csv[i])
        print(arr_cpu_time_csv[i])
        print(arr_disk_avg_csv[i])
        print(arr_disk_free_csv[i])
        print(arr_disk_queue_csv[i])
        print(arr_mem_available_csv[i])
        print(arr_mem_pages_csv[i])

        print(ex_events[i][1]/100000)
        print(ex_events[i][2]/100000)
        print(ex_events[i][3]/10)
        print(ex_events[i][4]/100)
        print(ex_events[i][5]/1000)
        print(ex_events[i][6]/10000)

        if len(total_list_call) > i:
            print(total_list_call[i][1]/100000)
        else:
            print(0)
        if len(total_list_excp) > i:
            print(total_list_excp[i][1])
        else:
            print(0)
        if len(total_list_dbmssql) > i:
            print(total_list_dbmssql[i][1]/100000)
        else:
            print(0)

        registry = CollectorRegistry()
        g = Gauge('perf_cpu_queue', 'Description metric', registry=registry)
        g.set(arr_cpu_queue_csv[i])
        push_to_gateway('vm-test-sea:9091', job='perf_cpu_queue', registry=registry)

        registry = CollectorRegistry()
        g = Gauge('perf_cpu_time', 'Description metric', registry=registry)
        g.set(arr_cpu_time_csv[i])
        push_to_gateway('vm-test-sea:9091', job='perf_cpu_time', registry=registry)
```

```
    registry = CollectorRegistry()
    g = Gauge('perf_disk_avg', 'Description metric', registry=registry)
    g.set(arr_disk_avg_csv[i])
    push_to_gateway('vm-test-sea:9091', job='perf_disk_avg', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('perf_disk_free', 'Description metric', registry=registry)
    g.set(arr_disk_free_csv[i])
    push_to_gateway('vm-test-sea:9091', job='perf_disk_free', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('perf_disk_queue', 'Description metric', registry=registry)
    g.set(arr_disk_queue_csv[i])
    push_to_gateway('vm-test-sea:9091', job='perf_disk_queue', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('perf_mem_available', 'Description metric', registry=registry)
    g.set(arr_mem_available_csv[i])
    push_to_gateway('vm-test-sea:9091', job='perf_mem_available', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('perf_mem_pages', 'Description metric', registry=registry)
    g.set(arr_mem_pages_csv[i])
    push_to_gateway('vm-test-sea:9091', job='perf_mem_pages', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('exev_duration', 'Description metric', registry=registry)
    g.set(ex_events[i][1]/100000)
    push_to_gateway('vm-test-sea:9091', job='exev_duration', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('exev_cpu_time', 'Description metric', registry=registry)
    g.set(ex_events[i][2]/100000)
    push_to_gateway('vm-test-sea:9091', job='exev_cpu_time', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('exev_physical_reads', 'Description metric', registry=registry)
    g.set(ex_events[i][3]/10)
    push_to_gateway('vm-test-sea:9091', job='exev_physical_reads', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('exev_logical_reads', 'Description metric', registry=registry)
    g.set(ex_events[i][4]/100)
    push_to_gateway('vm-test-sea:9091', job='exev_logical_reads', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('exev_writes', 'Description metric', registry=registry)
    g.set(ex_events[i][5]/1000)
    push_to_gateway('vm-test-sea:9091', job='exev_writes', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('exev_row_count', 'Description metric', registry=registry)
    g.set(ex_events[i][6]/10000)
    push_to_gateway('vm-test-sea:9091', job='exev_row_count', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('tj_call', 'Description metric', registry=registry)
    if len(total_list_call) > i:
        g.set(total_list_call[i][1]/100000)
    else:
        g.set(0)
    push_to_gateway('vm-test-sea:9091', job='tj_call', registry=registry)
```

```
    registry = CollectorRegistry()
    g = Gauge('tj_excp', 'Description metric', registry=registry)
    if len(total_list_excp) > i:
        g.set(total_list_excp[i][1]/100000)
    else:
        g.set(0)
    push_to_gateway('vm-test-sea:9091', job='tj_excp', registry=registry)

    registry = CollectorRegistry()
    g = Gauge('tj_dbmssql', 'Description metric', registry=registry)
    if len(total_list_dbmssql) > i:
        g.set(total_list_dbmssql[i][1]/100000)
    else:
        g.set(0)
    push_to_gateway('vm-test-sea:9091', job='tj_dbmssql', registry=registry)

    time.sleep(5)


push()
```

Пример настройки одной из панелей (Perfomance) дашборда Grafana показан на рисунке №25.

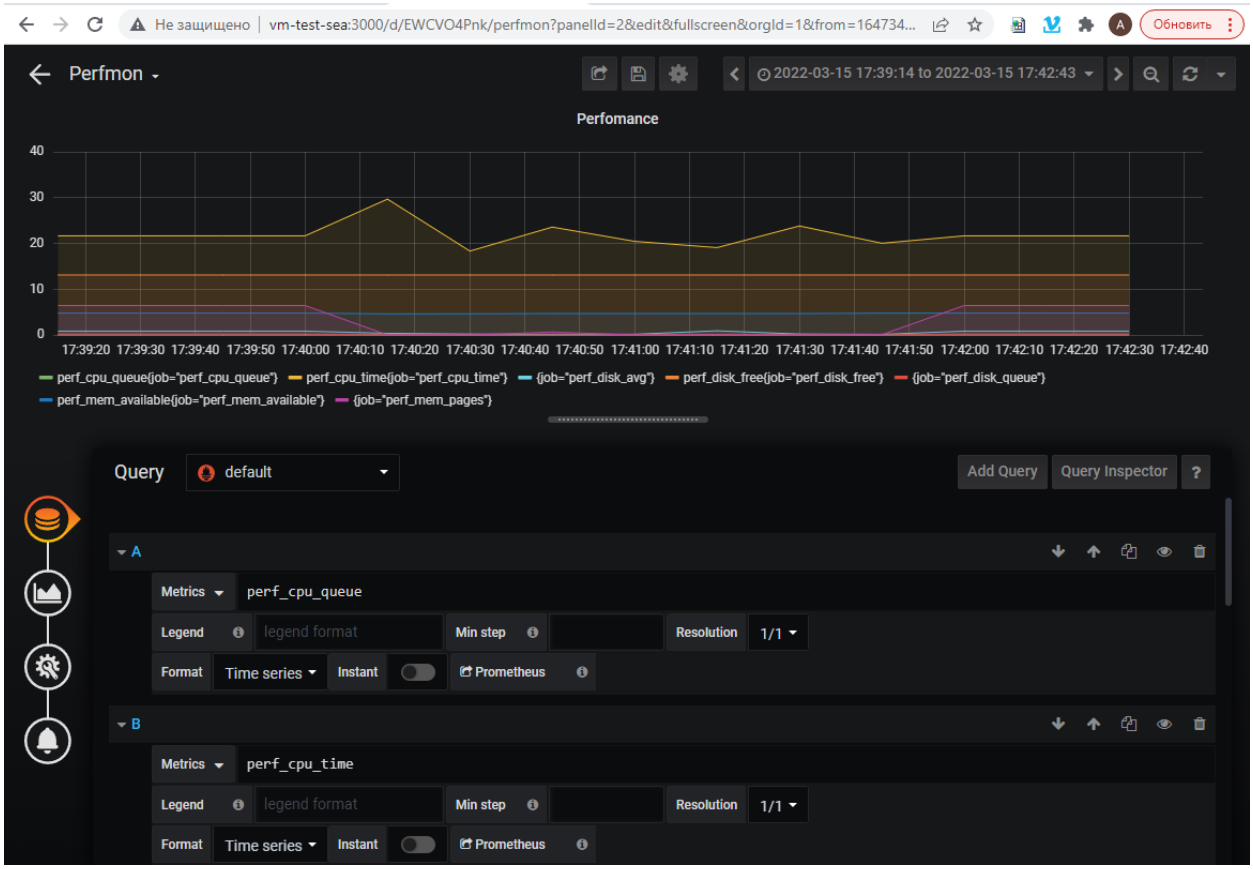


Рисунок №25. Настройки дашборда Grafana

Идея мониторинга в том, что, наблюдая за объединенным графиком показателей по Apdex, счетчику производительности, длинных запросов на сервере БД и длинных серверных вызовов на сервере приложения,

администратор системы сможет увидеть какие-то всплески, посмотреть подробности этих всплесков на соответствующих графиках и суметь принять решение по улучшению текущей производительности, чтобы работа пользователей не страдала и была комфортной.

На рисунке №26 дашборд с общим графиком и графиками, отражающими подробности работы каждой из частей продуктивного контура.

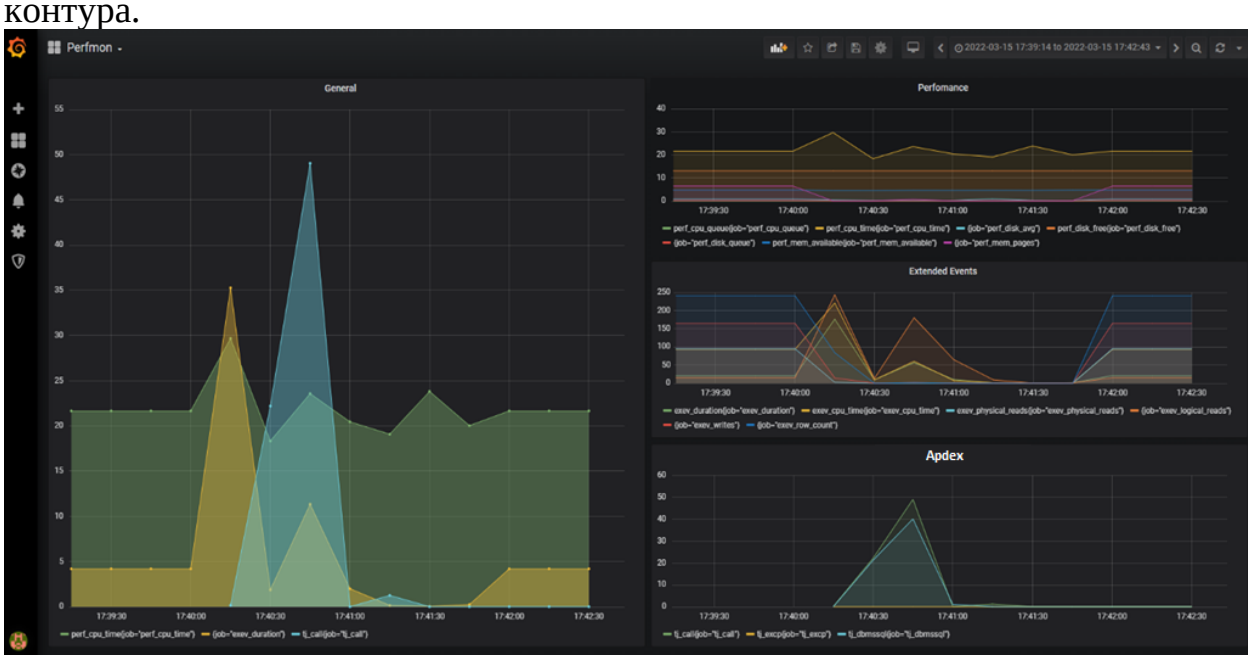


Рисунок №26. Итоговый дашборд

Структура модулей программного продукта

Для удобства отладки и доработки весь код, написанный на python, разбит на модули. На рисунке №27 показан список модулей – так как он хранится на жёстком диске.

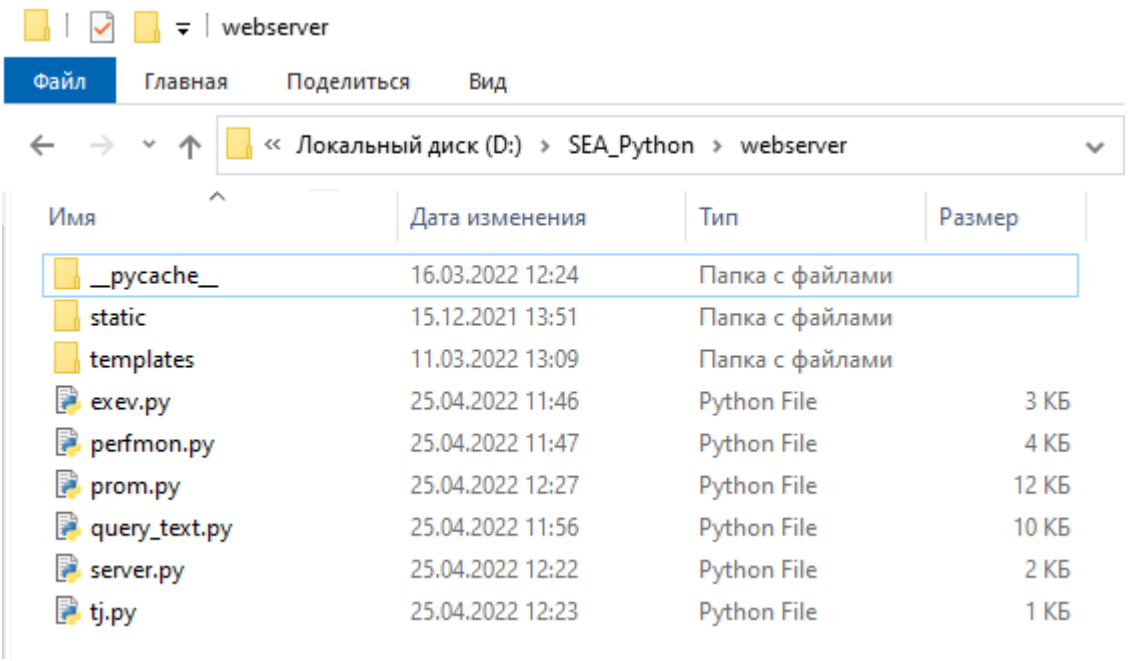


Рисунок №27. Модули

server.py – главный модуль, который запускает веб-страницу и работу остальных модулей.
exev.py – модуль процедур для работы с Extended Events
perfmon.py – модуль процедур для работы с счетчиком производительности оборудования
prom.py – модуль процедур для получения и отправки метрик из собранных логов в базу данных Prometheus
query_text.py – модуль процедур, возвращающих текст запросов для работы с Extended Events
tj.py – модуль процедур для работы с технологическим журналом 1С

Анализ производительности

Чаще всего виновниками низкой производительности являются неоптимально написанные запросы. В MS SQL при помощи расширенных событий (Extended Events) можно найти запросы, которые выполняются слишком долго. Или используют много процессорных ресурсов. Или нагружают диски. На это даже настроен один из дашбордов, который описан в разделе по мониторингу. Однако, что с таким запросом делать? Ведь он сформировался и передался на выполнение в MS SQL в приложении. В данной диссертации – в «1С:Предприятии 8.3». Как найти этот запрос в тексте кода приложения по тексту кода запроса MS SQL? Для этого можно использовать технологический журнал. Он может логировать и тексты запросов и контекст кода приложения, откуда этот запрос вызван. На python написан специальный скрипт, который производит сопоставление этих текстов. Алгоритм такой:

1. Все запросы из MS SQL группируются по тексту запроса и суммируются по нужному показателю. Например, по длительности
2. Производится сортировка по итоговой длительности и выбираются для анализа топ десять запросов
3. По тексту запросов находятся строчки в технологическом журнале, в которых есть контекст кода приложения

Алгоритм несложный. Однако есть несколько трудностей в его реализации. Во-первых, текст запроса нужно нормализовать. Имена временных таблиц заменить на какое-то одно имя (в разных сессиях один и тот же запрос может использовать разные имена временных таблиц). Во-вторых, логи технологического журнала для события DBMSSQL (а нужно именно оно) хранятся не в одной строке, а в нескольких. При этом контекст может быть вообще в другой строке. Для этого в алгоритме добавлено преобразование многострочного текста в одну строку. И для более быстрого поиска используются регулярные выражения.

Вот текст скрипта:

```
import pyodbc
import glob
import os
import re

def analyze_sql():

    connection_string = "Driver={SQL Server Native Client 11.0};" \
                    "Server= VM-TEST;" "Trusted_Connection=no;" \
                    "UID=sa;" "PWD=***;"
    db_conn = pyodbc.connect(connection_string, autocommit=True)
    cursor = db_conn.cursor()

    cursor.execute(get_top_10_sql())

    rows = cursor.fetchall()
    db_conn.close()

    list_of_lines = []
    for row in rows:
        write_log = False
        sql_text_lines = row.sql_text.splitlines()
        for line in sql_text_lines:
            if useful_line(line):
                write_log = True
                break

        if write_log:
            count_useful_line = 0
            line_for_tj = 'DBMSSQL,'
            for line in sql_text_lines:
                if useful_line(line):
                    count_useful_line += 1
                    line = re.sub(r'@P\d+', '?', line)
                    line = re.sub(r'\x00', '', line)
                    line_for_tj = line_for_tj + '.*' + line
            if count_useful_line > 3:
                list_of_lines.append(line_for_tj)
```

```
    return (list_of_lines)



def useful_line(line):
    useful = False
    if len(line) == 0 or line.find('>') > 0 or line.find('<') > 0:
        return False
    if line[0] == 'T':
        useful = True
    if len(line) > 9 and line[0:8] == 'FROM dbo':
        useful = True
    if len(line) > 4 and line[0:4] == 'CASE':
        useful = True
    if len(line) > 5 and line[0:5] == 'WHERE':
        useful = True
    if len(line) > 10 and line[0:10] == 'GROUP BY T':
        useful = True
    if len(line) > 12 and line[0:12] == 'INNER JOIN d':
        useful = True
    if len(line) > 17 and line[0:17] == 'LEFT OUTER JOIN d':
        useful = True
    return useful


def get_top_10_sql():
    sql = """
        select top 10
            xel.sql_text as sql_text,
            SUM(xel.duration) as duration,
            SUM(xel.cpu_time) as cpu_time,
            SUM(xel.physical_reads) as physical_reads,
            SUM(xel.logical_reads) as logical_reads,
            SUM(xel.writes) as writes,
            SUM(xel.row_count) as row_count
        FROM
            (select
                n.value('(@timestamp)[1]', 'datetime2') AS [utc_timestamp],
                n.value('(data[@name="duration"]/value)[1]', 'int') as duration,
                n.value('(data[@name="cpu_time"]/value)[1]', 'int') as cpu_time,
                n.value('(data[@name="physical_reads"]/value)[1]', 'int') as physical_reads,
                n.value('(data[@name="logical_reads"]/value)[1]', 'int') as logical_reads,
                n.value('(data[@name="writes"]/value)[1]', 'int') as writes,
                n.value('(data[@name="row_count"]/value)[1]', 'int') as row_count,
                dbo.fn_GetSQLNormalized(n.value('(action[@name="sql_text"]/value)[1]',
'nvarchar(max)')) as sql_text
            from (select cast(event_data as XML) as event_data
            from
sys.fn_xe_file_target_read_file('D:\SEA_TD_MON\SEA_ExEv\SEA_Query_Text_*.xel', null, null,
null)) ed
        cross apply ed.event_data.nodes('event') as q(n) ) as xel
        group by sql_text
        order by duration desc"""

    return sql


def analyze_tj():

    list_dbmssql = []

    for directory in glob.glob(r'D:\SEA_TD_MON\SEA_LOG\DBMSSQL\rphos*'):
```

```
        files = os.listdir(directory)
        for file_path in files:
            with open(directory + '\\' + file_path, encoding='utf-8') as file:
                line_dbmssql = ''
                for line in file:
                    if re.search(r'\d\d:\d\d\.\d.*?-\d.*,DBMSSQL,', line):
                        if line_dbmssql != '':
                            list_dbmssql.append(line_dbmssql)
                        line_dbmssql = line
                        line_dbmssql = re.sub(r'\r\n', '@NL', line_dbmssql)
                        line_dbmssql = re.sub(r'\n', '@NL', line_dbmssql)

                    elif re.search(r'\d\d:\d\d\.\d.*?-\d.*,Context,', line) and line_dbmssql != '':
                        line_dbmssql = line_dbmssql + re.sub(r'\d\d:\d\d\.\d.*?-\d.*,Context,',
",Context,", line)
                        line_dbmssql = re.sub(r'\r\n', '@NL', line_dbmssql)
                        line_dbmssql = re.sub(r'\n', '@NL', line_dbmssql)

                    elif line_dbmssql != '':
                        line_dbmssql = line_dbmssql + line
                        line_dbmssql = re.sub(r'\r\n', '@NL', line_dbmssql)
                        line_dbmssql = re.sub(r'\n', '@NL', line_dbmssql)

    print('analyze_sql.start')
    list_sql = analyze_sql()
    print('analyze_sql.end')
    for line_dbmssql in list_dbmssql:
        for line_sql in list_sql:
            line_sql = line_sql.replace(')', '\)')
            line_sql = line_sql.replace('(', '\(')
            line_sql = line_sql.replace('>', '\>')
            line_sql = line_sql.replace('<', '\<')
            line_sql = line_sql.replace('?', '\?')

            if re.search(line_sql, line_dbmssql):
                line_dbmssql = re.sub(r',DBMSSQL,.*,Context,', ',Context= ', line_dbmssql)
                line_dbmssql = re.sub('@NL', r'\n', line_dbmssql)
                print(line_dbmssql)


analyze_tj()
```

На рисунке №28 показан пример того, что этот скрипт выводит.

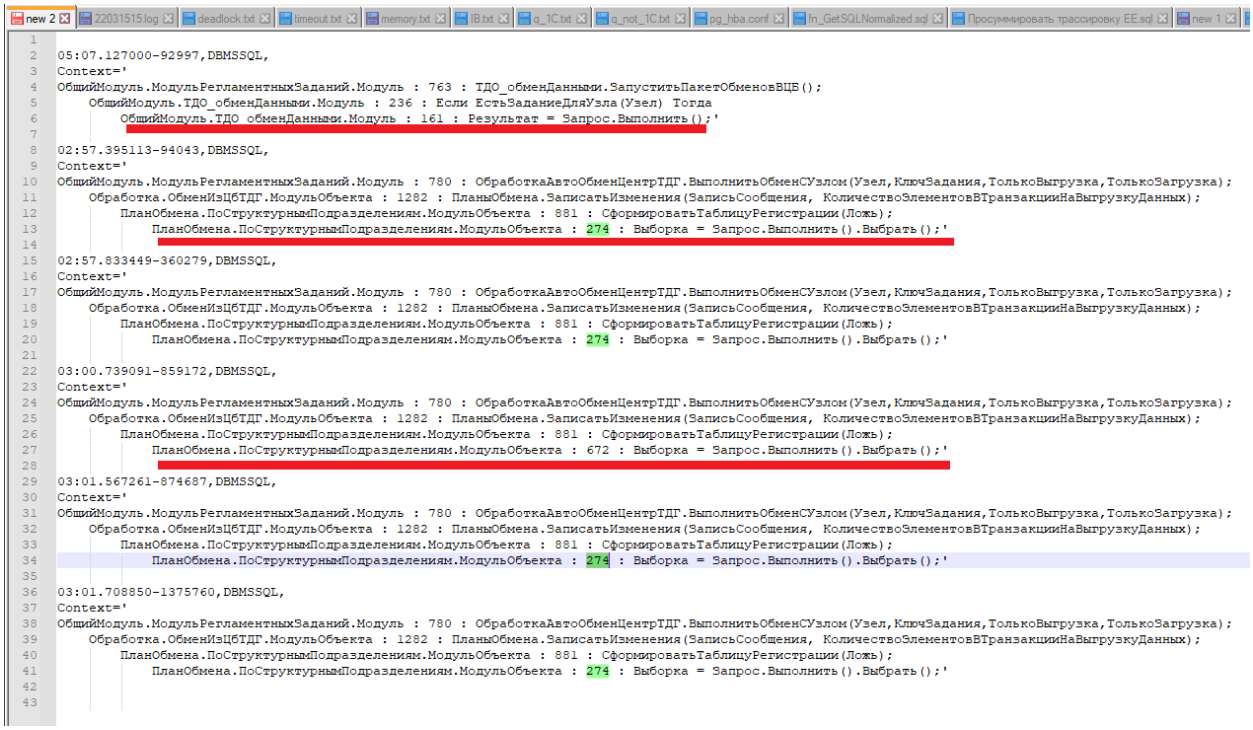


Рисунок №28. Анализ при помощи скрипта

Теперь нужно внимательно смотреть код приложения и пытаться понять, как можно его улучшить. Возможно, что стоит поменять логику. Добавить индексы в таблица на поля, по которым происходит выборка данных. Или пересмотреть архитектуру системы. Это уже работа творческая.

Дополнительно к анализу можно просматривать счетчики производительности. Иногда они подсказывают, почему запрос мог сработать плохо. В MS SQL есть механизмы рекомендации нехватающих индексов. Их анализ тоже может помочь. Но это уже более глубокий и пока неподдающийся автоматизации анализ.

Практическая польза

Исследование инструментов мониторинга производительности и использование визуализации дало свои плоды даже до окончания разработки инструмента по мониторингу и анализу производительности информационных систем. Ниже будут приведены два практических кейса, которые уже позволили улучшить работу продуктивного контура.

Практическое применение. Кейс №1

Параллельно разработке инструмента было решено при помощи коллег визуализировать ошибки, записывающиеся в журнал регистрации продуктивной системы. Когда информация об ошибках находится где-то в таблице, которую нужно открыть, поставить фильтры, сагрегировать количество, мало кто изъявляет желание заниматься этим, отвлекаясь от текущих задач. Однако, если вывести информацию об ошибках в виде графиков или гистограммы на телевизор, который видит весь оупенспейс, то сразу же находится куча желающих разобраться с аномальным количеством ошибок. Про одну из таких ошибок этот кейс.
На графике (рисунок №29) стало видно, что в узле распределённой базы данных «Центр» аномальное количество ошибок.

Рисунок №29. Много ошибок в узле «Центр»

Выгруженные из журнала регистрации тексты ошибок можно было увидеть в веб-интерфейсе инструмента «Elastic[19]» – рисунок №30.

19 https://www.elastic.co/

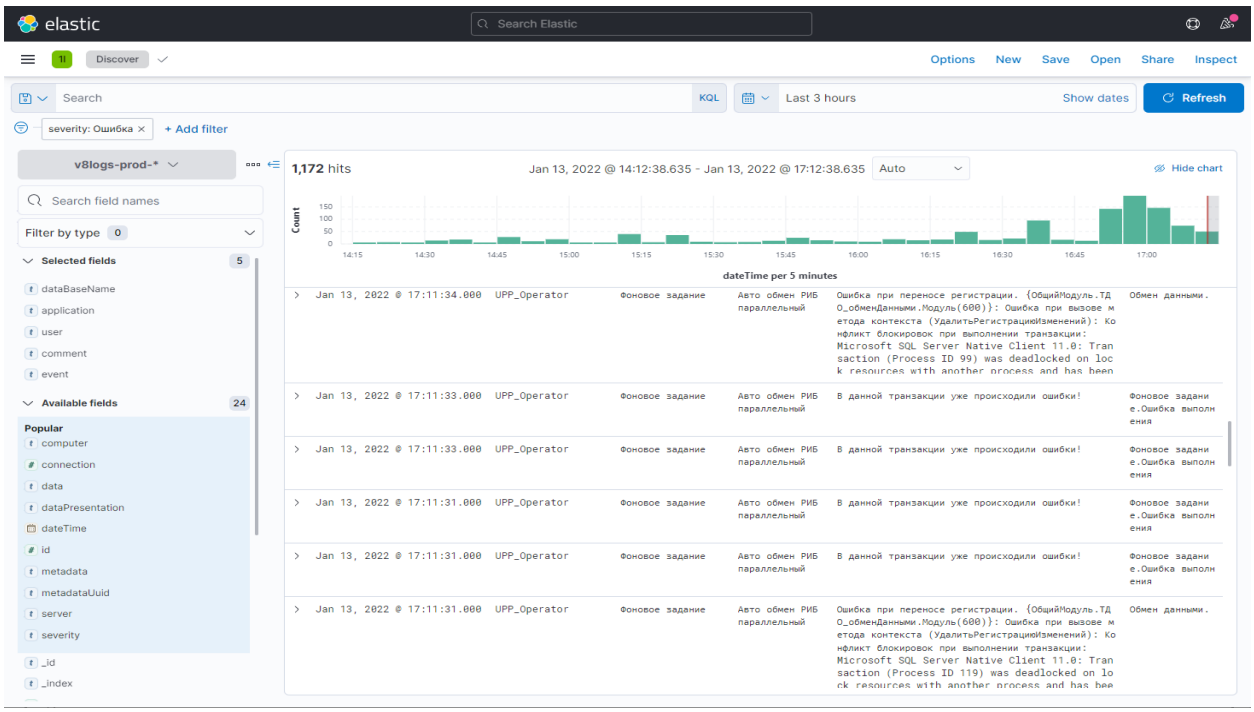


Рисунок №30. Тексты ошибок

При помощи механизма расширенных событий MS SQL удалось отловить конфликтующие запросы, приводящие к этим ошибкам. Это оказались взаимоблокировки, возникающие при регистрации данных для обмена одним сеансом и выгрузкой в файлы обмена этих данных другим сеансом. На рисунке №31 показана таблица выловленных запросов с графом взаимоблокировки одного из них.

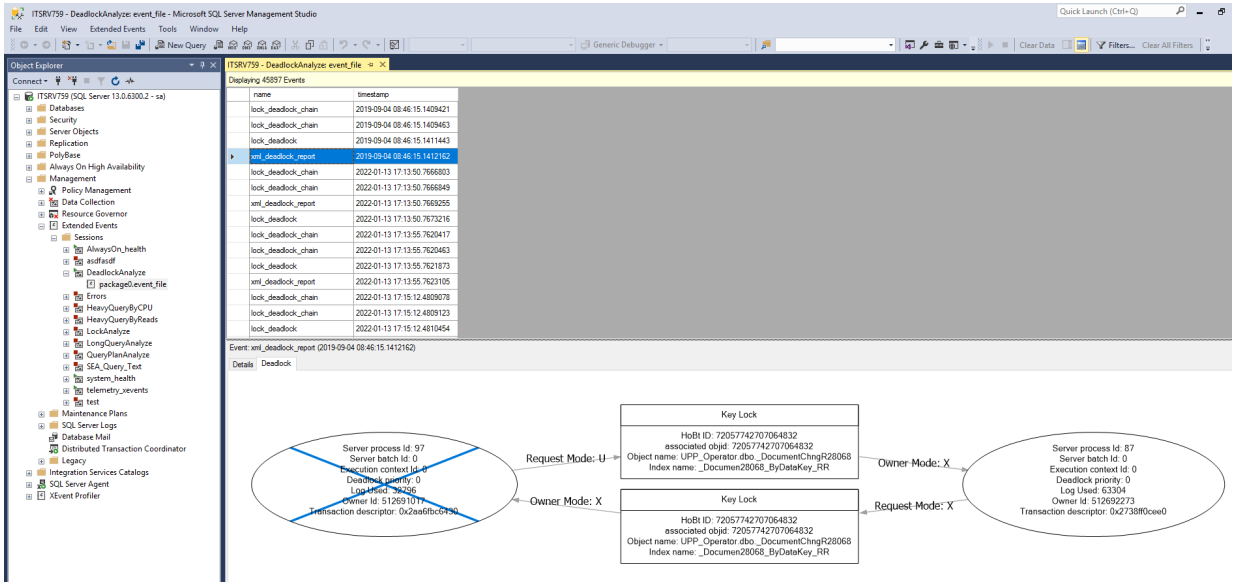


Рисунок №31. Запросы и граф взаимоблокировки

Расследовав более подробно этот граф (рисунок №32), удалось найти строчки кода, из которых вызывались конфликтующие запросы.

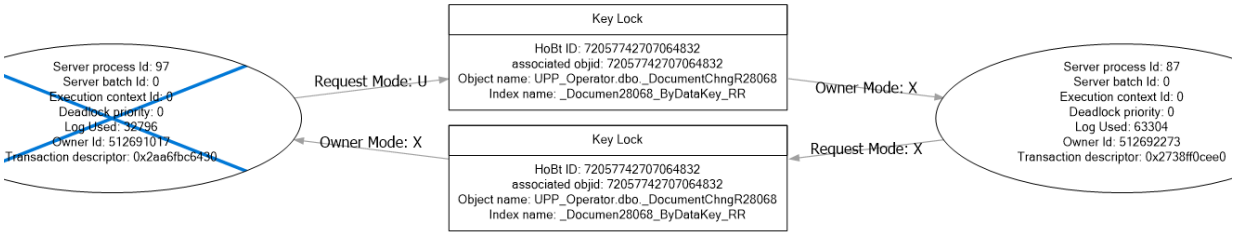


Рисунок №32. Граф взаимоблокировки. Крупно

Изменив немного алгоритм (на данные перед запросом были добавлены управляемые блокировки), удалось разрешить конфликт взаимоблокировок и добиться исчезновения связанных с ними ошибок.

Практическое применение. Кейс №2

В продуктивной информационной системе участились жалобы пользователей на периодическое подвисание. Отчет «Оценка производительности», основанный на настроенном в информационной системе Apdex, показывал, что периодически скачкообразно у пользователей увеличивалось время выполнения всех операций. Затем через несколько минут время выполнения возвращалось в норму. Анализ записей технологического журнала ничего нового не показывал. Также были видны провисания по времени выполнения операций. При помощи механизма MS SQL Extended Events удалось отловить топ запросов и увидеть, что есть запросы, читающие десятки тысяч строк данных из физической памяти в операционную (рисунок №33).

100 %

Results | Messages

	sql_text	username	database_name	duration	cpu_time	physical_reads	logical_reads	writes	row_count
1	(@P1 varbinary(16),@P2 varbinary(16))SELECT T1._IDRRef FROM dbo._Docum...	sa	UPP_Operator	874088130	868984000	0	72732745	0	0
2	(@P1 numeric(10),@P2 varbinary(16),@P3 datetime2(3),@P4 datetime2(3),@P5 n...	sa	UPP_Operator	680526689	715531000	262616	476983147	217	4
3	(@P1 varbinary(16))SELECT T1._Fld33408, T1._Fld33409RRef, T1._Fld33410, T...	sa	UPP_Operator	539712175	535837000	0	36947153	0	4
4	(@P1 varbinary(16))SELECT T1._Fld33408 FROM dbo._InfoRg33407 T1 WHER...	sa	UPP_Operator	536586667	531677000	0	36746032	0	0
5	(@P1 varbinary(16))SELECT T2._Fld23930RRef FROM dbo._Document414_VT2...	sa	UPP_Operator	523510230	517463000	0	17811069	0	542
6	(@P1 varchar(25),@P2 varchar(23))select top 50000 [Current LSN], [operation],...	dms_upp	UPP_Operator	475537340	448987000	174844	4299860	0	610654
7	(@P1 varchar(25),@P2 varchar(23))select top 50000 [Current LSN], [operation],...	dms_upp	UPP_Operator	463940005	446199000	174065	4279155	0	620273
8	(@P1 varchar(25),@P2 varchar(23))select top 50000 [Current LSN], [operation],...	dms_upp	UPP_Operator	462270393	440129000	168653	4304285	0	649999
9	(@P1 varchar(25),@P2 varchar(23))select top 50000 [Current LSN], [operation],...	dms_upp	UPP_Operator	453103197	445575000	174059	4319259	0	608141
10	(@P1 varbinary(16),@P2 varbinary(16))SELECT T1.Q_001_F_000TRef, T1.Q_00...	sa	UPP_Operator	31947447	317386000	5	15601582	0	9
11	(@P1 varbinary(16),@P2 varbinary(16))SELECT T1.Q_001_F_000TRef, T1.Q_00...	sa	UPP_Operator	312886670	311021000	0	17816597	0	6

Рисунок №33. Топ запросов

В графике замеров счетчиков производительности удалось подметить странное поведение счетчика «Page life expectancy» (рисунок №34). Этот счетчик показывает время жизни страниц памяти в оперативной памяти. Чем больше его значение, тем лучше.

Рисунок №34. Счетчик «Page life expectancy»

Разобравшись с тем, что могло означать резкое падение этого счетчика, удалось сделать вывод, что запросы, которые вычитывали много данных, «вымывали» из кэша (оперативной памяти) MS SQL все остальные страница памяти с планами запросов и результатами выполненных ранее более мелких запросов. Из-за этого «вымывания», пока кэш снова не заполнится популярными запросами, происходит подвисание.
Рекомендацией было: перенести регламент, который эти запросы запускал, на другое время. После переноса это регламента, согласно рекомендациям, проблема исчезла.

Заключение

В заключении можно сказать, что итоговые выводы по степени важности расходятся с первоначальными ожиданиями. Основная идея и новшество инструмента, который разрабатывался в рамках данной диссертации, было размещение на едином графике не только показателей загруженности ПО и «железа», но и общего «самочувствия» информационной системы, выраженное через индекс производительности – Apdex. Это должно было направить специалиста, собирающегося производить какие-то действия по улучшению работы продуктивной системы, на более ощутимые места для пользователей. Поставить в приоритет именно те задачи, которые дадут для них наибольший эффект. На деле же оказалось, что не менее важным является наглядная визуализация данных мониторинга. Исправляются быстрее те ошибки, которые находятся перед глазами. Вся автоматизация в конечном счете всегда сводится к человеку – к специалисту, который принимает итоговое решение. И наглядность, которую даёт визуализация, запросто выигрывает у текста и цифр анализа. Быстрее исправляется то, что горит красным цветом перед лицом специалиста, чем то, что где-то в текстовом файле записывается автоматическим анализатором.

Однако, этот вывод не мешает констатировать тот факт, что поставленные задачи выполнены. Инструмент для мониторинга и анализа производительности информационной системы разработан. В дальнейшем он обязательно будет использоваться в продуктивном контуре компании АО «Технодом Оператор». При необходимости дорабатываться, улучшаться, изменяться. То есть, очевидна и исследовательская польза, полученная в результате реализации диссертации, и практическая.

Список использованных источников

В процессе исследования и написания магистерской диссертации была использована следующая литература:

1. А. Морозов, В. Федоров, А. Асатрян, А. Голиков, Д. Соломатин «Методическое пособие по эксплуатации крупных информационных систем на платформе «1С:Предприятие 8»»
2. Александр Бондарь «Microsoft SQL Server 2014»
3. Алексей Береснев «Администрирование GNU/Linux с нуля»
4. Бен Форта «Регулярные выражения. 10 минут на урок»
5. Джефри Фридл «Регулярные выражения»
6. Джон Мак-Кейб «Введение в Windows Server 2016»
7. Душан Петкович «Microsoft SQL Server 2012. Руководство для начинающих»
8. Войтов Никита Михайлович «Администрирование Red Hat Enterprise Linux»
9. Е. Филлипов «Настольная книга 1С:Эксперта по технологическим вопросам»
10. Ицик Бен-Ган «Microsoft SQL Server 2012. Основы T-SQL»
11. М. Г. Радченко, Е. Ю. Хрусталева «Архитектура и работа с данными «1С:Предприятия 8.2»
12. Моргунов Е. П. «PostgreSQL. Основы языка SQL: учеб. пособие»
13. Руководство администратора «1С:Предприятие 8.3»
14. Скотт Граннеман «Linux. Карманный справочник»
15. П. Лузанов, Е. Рогов, И. Лёвшин «Postgres: первое знакомство»
16. Уильям Станек «Microsoft Windows Server 2012. Справочник администратора»
17. Эрик Редмонд, Джим Р. Уилсон «Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL»
18. ZABBIX 6.0 LTS. Универсальный мониторинг – https://www.zabbix.com/ru/
19. PRTG – https://www.ru.paessler.com/
20. Центр управления производительностью – https://v8.1c.ru/tekhnologii/tekhnologii-krupnykh-vnedreniy/korporativnyy-instrumentalnyy-paket/tsentr-upravleniya-proizvoditelnostyu
21. Центр контроля качества – https://v8.1c.ru/tekhnologii/tekhnologii-krupnykh-vnedreniy/korporativnyy-instrumentalnyy-paket/tsentr-kontrolya-kachestva
22. https://infostart.ru/1c/articles/1040073
23. http://www.gilev.ru
24. Grafana – https://grafana.com
25. Установка и настройка Prometheus – https://losst.ru/ustanovka-i-nastrojka-prometheus
26. Flask – https://flask.palletsprojects.com/en/2.0.x/quickstart/

27. Extended events – https://docs.microsoft.com/ru-ru/sql/relational-databases/extended-events/extended-events?view=sql-server-2017