МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Системы автоматического управления»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
д.т.н., профессор
_____/ В.И. Ширяев
«____» _____ 2019 г.

Система мониторинга состояния аппаратной части узлов суперкомпьютера "Торнадо ЮУрГУ"

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ – 09.03.01.2019.134.00 ПЗ ВКР

Руководитель работы
доцент каф. САУ, к.т.н.
_____/ В.О. Чернецкий
«____» _____ 2019 г.

Автор работы
студент группы КЭ-451
_____/ О.М. Попонин
«____» _____ 2019 г.

Нормоконтролер
доцент каф. САУ, к.т.н.
_____/ В.О. Чернецкий
«____» _____ 2019 г.

Челябинск 2019

АННОТАЦИЯ

Попонин О.М. Система мониторинга состояния аппаратной части узлов суперкомпьютера "Торнадо ЮУрГУ". – Челябинск: ЮУрГУ, ВШ ЭКН; 2019, 37 с., 28 ил., библиогр. список – 18 наим., 11 листов слайдов презентации ф.А4, 1 приложение.

В рамках выпускной работы разработана и реализована система мониторинга состояния аппаратной части узлов суперкомпьютера "Торнадо ЮУрГУ".

В первой главе выполнен анализ основных принципов в разработке систем мониторинга аппаратных комплексов и суперкомпьютеров.

Во второй главе выполнен обзор ссуществующих систем мониторинга аппаратных комплексов и выявлены их преимущества и недостатки. Были изучены основные способы опроса узлов суперкомпьютера.

В третьей главе описаны требования к программе, выполнено проектирование главных его функций и определены варианты использования.

В четвертой главе представлены основные функции системы мониторинга, приведены примеры и описаны некоторые методы взаимодействия системы мониторинга с базой данных.

Система мониторинга состояния аппаратной части узлов суперкомпьютера "Торнадо ЮУрГУ" тестируется с целью добавление мониторинга состояния инфокоммуникационной сети суперкомпьютера.

Пояснительная записка к выпускной квалификационной работе оформлена в текстовом редакторе MS Word 2016.

					09.03.01.2019.134.00 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Попонин О.М.			Система мониторинга состояния аппаратной части узлов суперкомпьютера "Торнадо ЮУрГУ"	Лит.	Лист 4 / Листов 37
Провер.		Чернецкий В.О.				Д	
Н. Контр.		Чернецкий В.О.				ЮУрГУ Кафедра САУ	
Утверд.		Ширяев В.И.					

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данная работа посвящена разработке системы мониторинга состояния аппаратной части узлов суперкомпьютера «Торнадо ЮУрГУ» для суперкомпьютерного центра Лаборатории суперкомпьютерного моделирования ЮУрГУ. В настоящее время в связи с развитием информационных технологий возрастает потребность в вычислительных ресурсах. Примером такого роста может служить рейтинг Топ-500 мощнейших суперкомпьютеров мира [1]. С увеличением количества вычислительных узлов в суперкомпьютерах возрастает потребность в системах, способных упростить задачи администрирования и мониторинга.

Для стабильной работы суперкомпьютера требуется безотказность всех составляющих компонентов. Так как суперкомпьютер постоянно находится в работе, необходимо следить, чтобы все его узлы находились исправном аппаратном состоянии и имели не превышающие норму показания по датчикам температуры центрального процессора, напряжения по линиям питания 5 и 12 вольт, напряжения элемента питания CMOS.

Для того чтобы отслеживать аппаратное состояние узлов и понимать, когда необходимо их обслуживать требуется создать систему мониторинга аппаратного состояния узлов суперкомпьютера.

Актуальность темы

Системы мониторинга суперкомпьютерных комплексов – важная часть их программного обеспечения, призванная обеспечить максимально эффективное и бесперебойное функционирование, а в случае возникновения нештатных ситуаций – сохранность оборудования и оповещение администраторов.

Существующие системы мониторинга такие как Zabbix [2], Nagios [3], Supermicro IPMIView [4] не способны учитывать все особенности проектирования суперкомпьютеров и отслеживать состояния суперкомпьютера в целом. И, как правило, задействуются только на основных (головных) узлах суперкомпьютера.

На суперкомпьютере “Торнадо” ежемесячно считается большое количество задач, требующих высокой производительности суперкомпьютера. Задачи распределяются среди 480 узлов суперкомпьютера. Если во время выполнения задачи один из участвующих в расчете узлов выйдет из строя — это гарантированно прервет решение задачи на остальных узлах, а задача будет запущена заново. Повторный запуск прерванной задачи снижает эффективность использования рабочего времени суперкомпьютера, а также замедляет выполнение других задач.

Каждая задача задействует различное множество узлов, вплоть до максимального количества узлов суперкомпьютера – 480.

К основным причинам выхода из строя узла суперкомпьютера относятся:

- перегрев модулей оперативной памяти;
- высыхания термопасты между центральным процессором и жидкостной системой охлаждения;
- выход из строя элементов питания центрального процессора, что приводит к перенапряжению всей системы питания;
- элемент питания CMOS, который отработал свой ресурс и полежит замене для полноценного функционирования узла.

В статье «современные суперкомпьютеры» журнала «сетевые решения/LAN» [5] был исследованы Топ 5 суперкомпьютеров за 2013й год и были представлены эксплуатационные нормы по основным показаниям узлов суперкомпьютеров, которые указаны в таблице 1.1

Таблица 1.1 – Эксплуатационные нормы показаний датчиков

Датчик	**Эксплуатационная норма**
Температура процессора	40-60 ℃
Температура модуля оперативной памяти	40-70 ℃
Напряжение по линии 12 вольт	11.8 – 12.6 В
Напряжение по линии 5 вольт	4.8 – 5.2 В
Напряжение элемента питания CMOS	3 – 3.3 В

Для автоматического выявления данных ситуаций необходимо организовывать опрос (активный мониторинг) датчиков каждого узла суперкомпьютера с достаточно малым периодом времени, например, раз в минуту. При этом необходимо обеспечить высокую надежность обнаружения аварийных узлов, так как задержка или отсутствие обслуживания могут привести к выходу из строя большого количества дорогостоящего оборудования. Поток данных, который нужно обрабатывать, небольшой, поскольку количество инфраструктурного оборудования и датчиков в нем невелико даже в больших комплексах.

Предложенная в данной работе система позволяет администраторам суперкомпьютерного центра ЮУрГУ отслеживать состояние аппаратной части узлов суперкомпьютера «Торнадо ЮУрГУ» и выявлять узлы, которым требуется обслуживание.

Цель и задачи исследования

Целью данной работы является разработка программы мониторинга состояния аппаратной части узлов суперкомпьютера “Торнадо ЮУрГУ”

Для достижения поставленной цели, необходимо решить следующие ***задачи***:

1) Выполнить анализ предметной области и обзор существующих систем мониторинга;

2) Спроектировать программную систему мониторинга аппаратного состояния узлов суперкомпьютера;

3) Реализовать спроектированную программную систему.

Структура и объем работы

Выпускная квалификационная работа состоит из введения, четырех разделов, заключения и библиографии. Объем работы составляет 52 страницы, объем библиографии – 16 источников.

Содержание работы

В главе «Анализ предметной области» выполнен анализ основных принципов в разработке систем мониторинга аппаратных комплексов и суперкомпьютеров.

В главе «Обзор существующих решений» выполнен обзор ссуществующих систем мониторинга аппаратных комплексов и выявлены их преимущества и недостатки. Были изучены основные способы опроса узлов суперкомпьютера.

В главе «Проектирование» описаны требования к программе, выполнено проектирование главных его функций и определены варианты использования.

В главе «Разработка программной системы» представлены основные функции системы мониторинга, приведены примеры и описаны некоторые методы взаимодействия системы мониторинга с базой данных.

В заключении описываются основные результаты, полученные при выполнении дипломной работы.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данной главе описаны существующие принципы в разработке топологии и архитектуре систем мониторинга датчиков аппаратных комплексов и суперкомпьютеров.

1.1 Архитектура систем мониторинга.

На рисунке 1.1 изображена общая архитектура систем мониторинга. За получение информации отвечают агенты, запущенные на вычислительны узлах. Некоторые системы характеризуются как безагентные [6]. В этом случае роль агентов играют сервисы операционной системы, работающие на узле. Это могут быть сервера SNMP или доступ по протоколу SSH для получения информации. При получении информации о состоянии других компонент вычислительной системы вместо агентов на вычислительном узле используются агенты SNMP или компоненты, отвечающие за получение данных по каким то другим протоколам. Собранные таким способом данные передаются в серверную часть системы мониторинга, где происходит их обработка, сохранения части информации в базу данных, выделение событий, реагирование на события и имеются части, предназначенные для представления данных пользователю, в частности визуализации.

Значительное количество существующих систем мониторинга [6-12] построено по описанной схеме. Их общие особенности описаны далее.

Во-первых, это отсутствие какой бы то ни было обработки данных на вычислительном узле, являющемся источником этих данных. Причины этого очевидны для безагентных систем: высокая эффективность работы узла и малый объем данных. Но и в системах, у которых имеются на вычислительных узлах собственные агенты, эти агенты только собирают необходимую информацию и передают ее дальше без обработки. Обычно такой подход объясняют необходимостью уменьшить влияние работы агента на ход выполнения вычислительной задачи, которая работает на этом вычислительном узле.

Другой особенностью существующих систем мониторинга является жесткая конфигурация путей передачи данных. Пути передачи данных задаются при конфигурировании системы и не меняются в процессе работы. Для каждого агента задается единственный адрес в серверной части системы мониторинга (или несколько адресов, если серверная часть строится с дублированием), куда пересылаются собираемые данные, и этот адрес не меняется в процессе работы.

Для обеспечения необходимой производительности серверную часть системы мониторинга делают распределенной с древовидной структурой. Каждый листовой компонент серверной части отвечает за какую-то часть вычислительных узлов. В этом компоненте информация обрабатывается, агрегируется и выделяются

события. Затем обработанный и уменьшенный поток информации передается на следующие уровни для дальнейшей обработки.

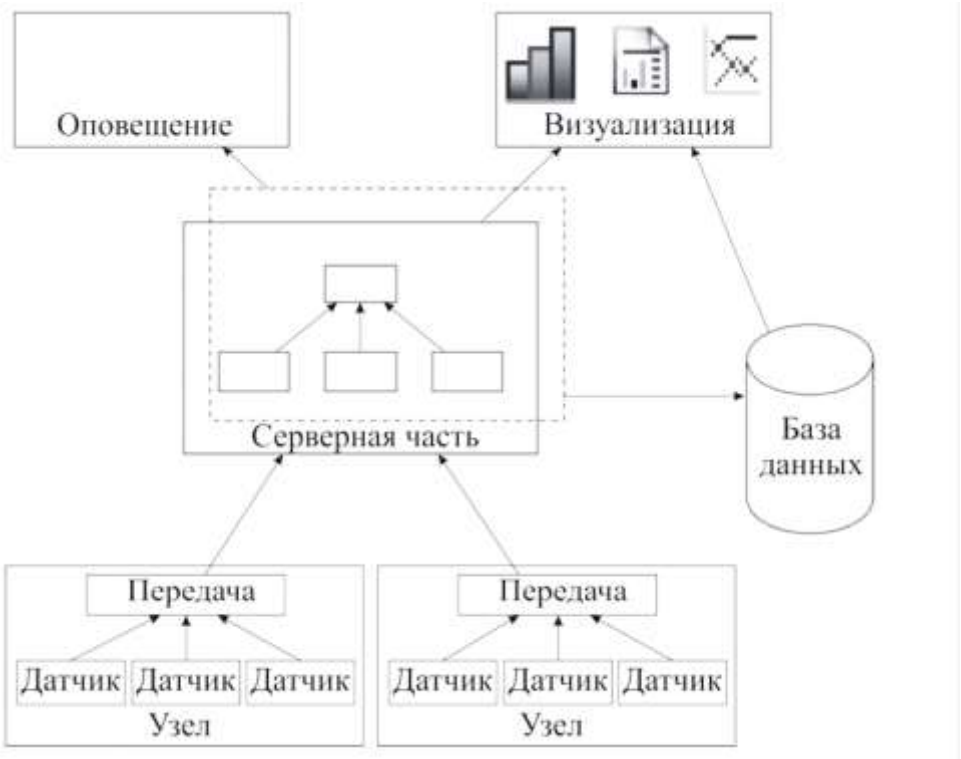

Рисунок 1.1 – Общая архитектура систем мониторинга

Распределенная система мониторинга

Распределенная система мониторинга состоит из агентов мониторинга. Агент мониторинга с точки зрения операционной системы – это процесс, работающий на вычислительном узле или на другом компьютере (сервере). Агент мониторинга представляет собой систему исполнения – программное окружение, которое обеспечивает функционирование узлов, связей между ними и передачу сообщений.

Всё получение и обработка информации о состоянии вычислительной системы ведется в вычислительных узлах суперкомпьютера. Данные передаются от узла к узлу. При этом каждый узел конфигурируется независимо от других. Связи между узлами создаются независимо друг от друга.

В системе передаются сообщения двух типов. Основной тип- сообщения, содержащие данные мониторинга. Они передаются от узла к узлу вдоль созданных связей между узлами. Второй тип – управляющие сообщения, они передаются непосредственно от одного узла к другому вне зависимости от наличия между ними связей.

Каждый узел имеет ноль или больше именованных входов, ноль или больше именованных выходов и может принимать и передавать управляющие сообщения.

Все узлы имеют числовой идентификатор, присеваемый системой исполнения при их создании. Кроме того, узлу может быть присвоено имя (строка символов)

При создании связи между узлами задаются узел – источник данных, имя выхода этого узла для создания связи, узел – приемник данных и имя входа, на которые данные будут передаваться. После создания связи копия данных, которые узел будет передавать на данный выход, будет передаваться системой исполнения на все выходы, связанные с данным входом

Управляющие сообщения служат для создания и уничтожения связи узлов, задания их имен, создания таймеров – такие сообщения обрабатываются системой исполнения (агентом мониторинга). Сообщения, служащие для изменения настроек узлов, и другие, специфичные для конкретного узла (типа узлов), обрабатываются непосредственно узлами, которым они адресованы. Управляющее сообщение может быть послано любым узлом любому другому. Для определения получателя сообщения используется имя или номер узла –получателя

Для построения системы мониторинга используется несколько видов узлов. Они отличаются лишь с точки зрения их функциональности, системой исполнения все узлы обрабатываются одинаково.

Узел, который получает данные мониторинга от программно-аппаратной среды (операционной системы или оборудования), называется датчиком. Датчик – это узел без входов. Его назначение – по сигналу от таймера получить данные мониторинга путем обращения к аппаратуре, сервисам ОС и т.п. и передать его на свой выход.

Узел с входами и выходами осуществляет обработку данных мониторинга. Обработка может быть любой: фильтрация данных, вычисление скорости изменения величины, сравнение с заданным порогом и т.д. Именование входов и выходов нужно для разделения потоков данных. Например, узел, отбирающий данные по какому-то критерию, может передавать данные, подходящие под критерий фильтрации, а данные, не подходящие под критерий – отбрасывать. Соединив эти выходы с другими узлами, мы получаем разделение потоков данных, которые затем могут обрабатываться независимо. Или же имя входа может определять типы данных мониторинга, которые будут передаваться.

Узлы с входами, но без выходов предназначены для передачи данных мониторинга вне агента. Отсутствие выходов с точки зрения системы исполнения, а не выход для передачи информации вообще. Чаще всего такой узел будет использоваться для организации взаимодействия агентов. Узел принимает данные мониторинга на своих входах, обрабатывает их и передает по сети другому агенту, работающему на другом компьютере или на том же самом. На входе принимающего агента будет работать узел-приемник, который во многом устроен как датчик, од-

нако данные он получает не от аппаратуры или ОС, а принимает по сети от другого агента. Настройка приемников и передатчиков осуществляется путем посылки управляющих сообщений.

Архитектура агента системы мониторинга, состоящей из двух датчиков, узла консолидации и узла для передачи данных по сети, с конфигурацией получения данных и передача их в серверную часть показана на рисунке 1.2.

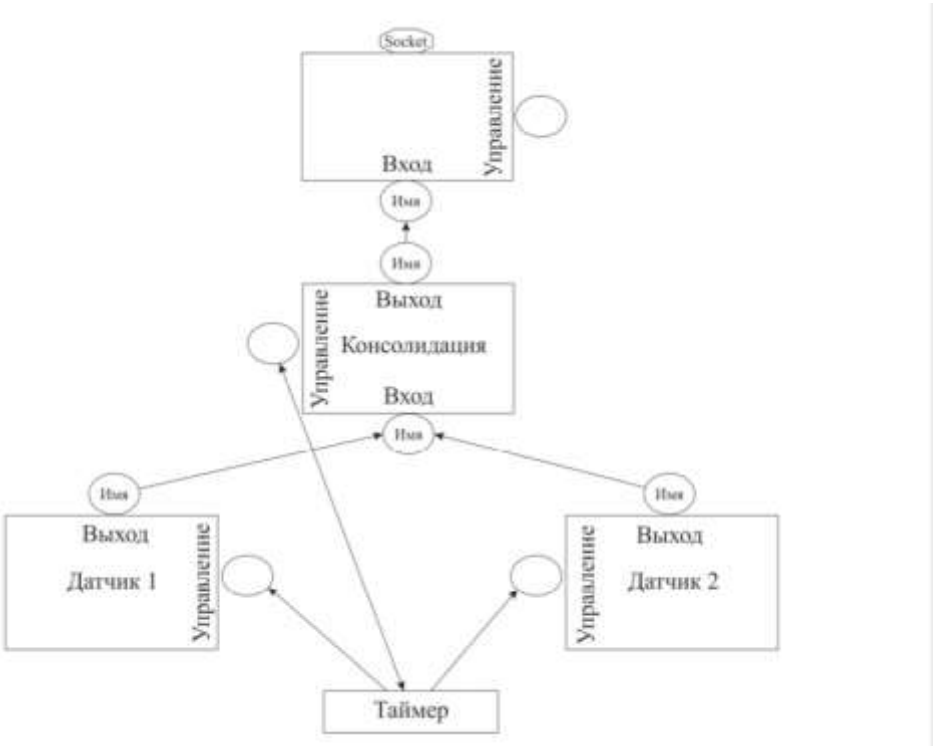

Рисунок 1.2 – Архитектура системы с двумя датчиками

Другой вариант конфигурации, показан на рисунке 1.3. В той конфигурации показаны три цели обработки данных мониторинга.

Первая цель требует высокой надежности обработки так-как это узел отслеживания аварийных ситуаций. Реализующие ее агенты дублированы на двух независимых серверах, имеют узлы для слежения друг за другом. Агент, реализующий реакции, не дублирован, но может работать по сигналу от любого из дублей нижележащей части.

Вторая цель – построение общесистемных метрик. Для этого используются датчики, отличные от тех, что использованы в первой цели. Соответствующие данные идут на один выделенный сервер для обработки.

Наконец третья цель – анализ отдельных задач. Для этого запускаются отдельные агенты (или узлы внутри агентов) на каждую задачу, на которые поступают данные от вычислительных узлов, на которых запущена данная задача.

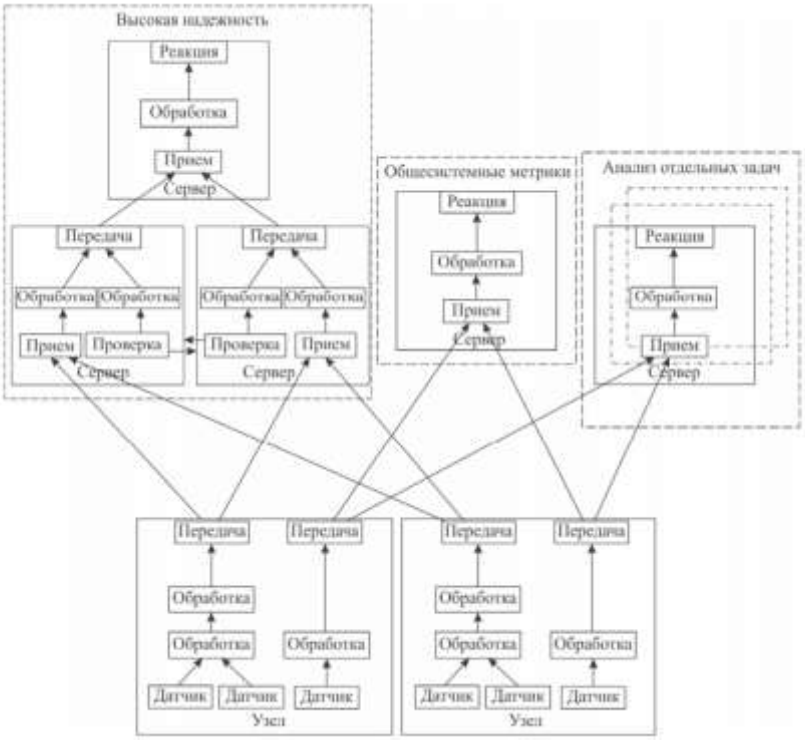

Рисунок 1.3 – Архитектура системы с тремя целями обработки

Оценки производительности данной системы показали, что скорость приема и передачи данных может достигать 350 тыс. пакетов в секунду, что составит примерно 230 Мбит/с. В проверке возможности обрабатывать данные от многих источников средняя нагрузка на серверный процессор Intel Xeon X5670x, работающего на частоте 2,93 ГГц достигала 12 %, что составляет более десятой всей производительности узла.

Централизованная система мониторинга

В централизованной системе мониторинга на узлах не используются агенты (сервисы). Сбором показаний с датчиков занимается только один узел, называемый сервером. Это позволяет не тратить вычислительную мощность рабочих узлов и не нарушать процесс расчетов задач на узле.

Централизованная система мониторинга состоит из двух основных элементов: сервера, который собирает показания с датчиков, и вычислительных узлов, показания которых необходимо отслеживать.

Сервер с помощью SNMP или SSH протокола опрашивает каждый узел отдельно и записывает все показания с датчиков в свою локальную базу данных, затем анализирует ее и ведет статистику.

Архитектура Централизованной системы мониторинга изображена на рисунке 1.4.

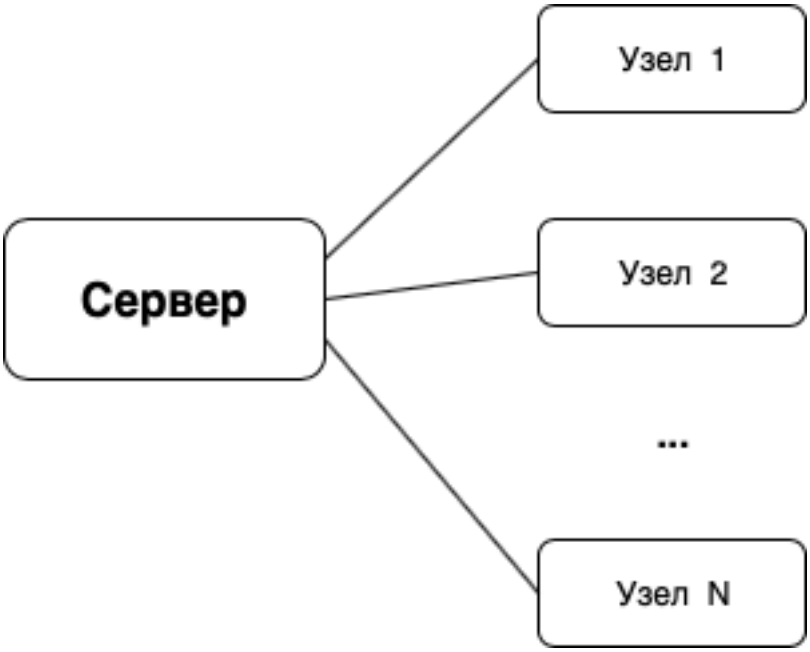

Рисунок 1.4 – Централизованная система мониторинга

В Централизованной системе мониторинга узлов суперкомпьютера важна стабильность работы сервера, так-как его поломка – приведет к прекращению мониторинга узлов. Поэтому при выборе сервера должны учитываться повышенные требования к его стабильности работы. Как правило, такие сервера устанавливаются в отдельную стойку и подключаются к отдельной линии питания, чтобы работа связующего узла никогда не прекращалась.

Так как в централизованной системе мониторинга все данные хранятся на локальном сервере – накапливается большой объем данных. Важно, чтобы у сервера была возможность дублировать базу данных с показаниями на сторонние хранилища для обеспечения сохранности всей системы.

Большим преимуществом перед распределенной системой мониторинга является гораздо меньшее использование сетевого трафика суперкомпьютера и минимальные затраты процессорного времени на чтение и запись в базу данных. средняя нагрузка на серверный процессор Intel Xeon X5670x, работающего на частоте 2,93 ГГц достигала 3 % всей производительности узла.

1.2 Обзор существующих решений

В данной главе описаны существующие системы с открытым исходным кодом, предназначенные для мониторинга аппаратных комплексов и суперкомпьютеров.

Система мониторинга Zabbix

Zabbix – это бесплатная система мониторинга отслеживания статусов разнообразных сервисов компьютерной сети, серверов и сетевого оборудования. Zabbix является решением с открытым исходным кодом, которое может производить комплексный мониторинг инфраструктуры, визуализировать полученную информации в графики, следить за нагрузкой и производительностью оборудования с использованием собственных агентов, поддерживается всеми операционными системами. Недостатком такой системы является агент – демон, который необходимо устанавливать и запускать на отслеживаемых объектах. Зачастую для решения различных вычислительных задач необходимы различные версии одной операционной системы, но имеющие разный набор программного обеспечения, т.к. некоторые программные пакеты могут быть несовместимы друг с другом. Пример интерфейса системы Zabbix представлен на рисунке 1.5

Рисунок 1.5 – Интерфейс системы мониторинга Zabbix

Интеллектуальный интерфейс Sepermicro IPMIView

Интеллектуальный интерфейс управления платформой, предназначенный для автономного мониторинга и управления функциями, встроенными непосредственно в аппаратное и микропрограммное обеспечения серверных платформ. Ключевые характеристики Supermicro IPMIView мониторинг, восстановление функций управления, журналирование и инвентаризация, которые доступны неза-

висимо от процессора, BIOS'a и операционной системы. Функции управления платформой могут быть доступны, даже если система находится в выключенном состоянии. Недостатком такой системы является отсутствие общей базы данных и уведомлений пользователя. Каждый узел придется проверять самостоятельно. Пример интерфейса системы Supermicro IPMIVIev представлен на рисунке 1.6

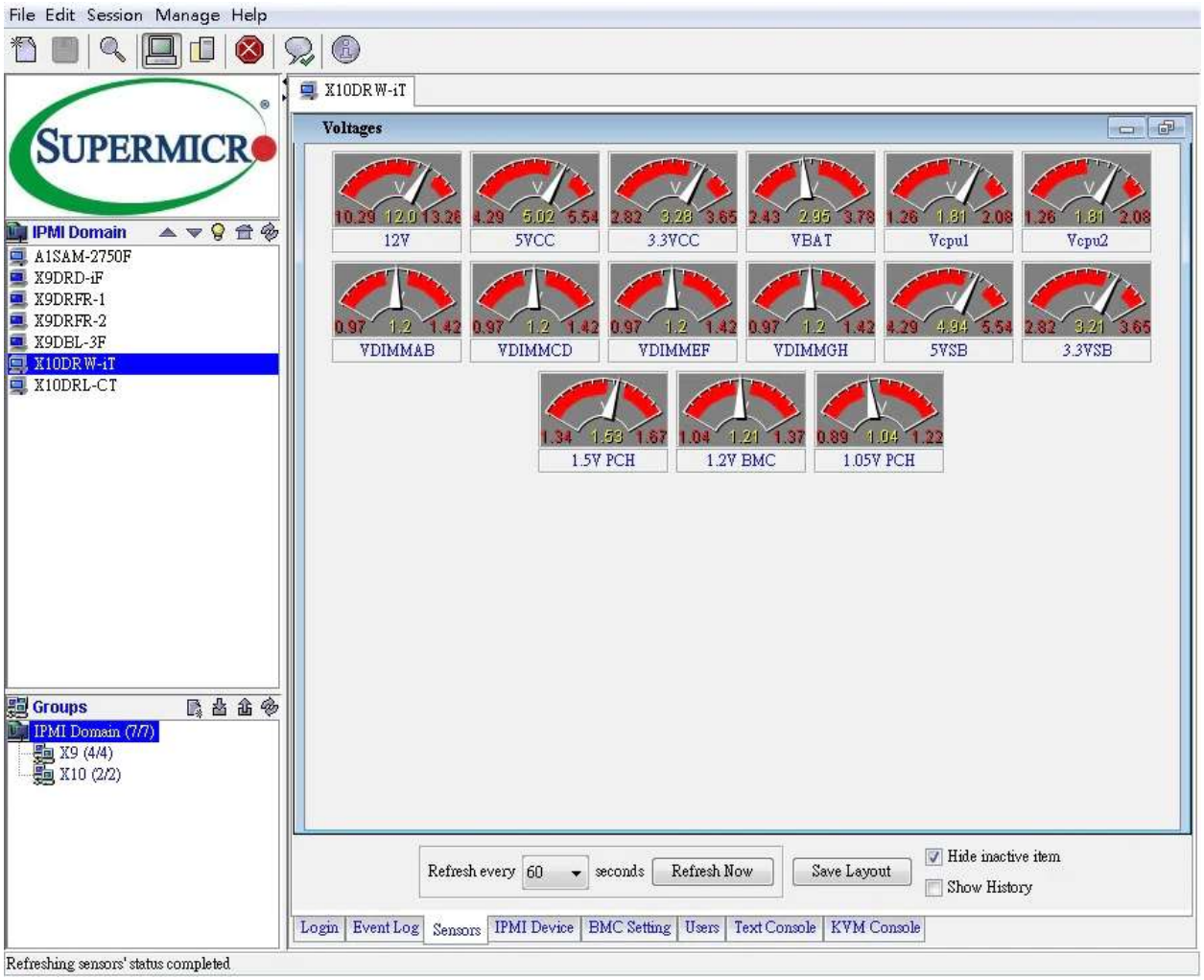

Рисунок 1.6 – Интерфейс системы мониторинга Supermicro IPMIView

Система мониторинга Nagios

Промышленный стандарт в мониторинге ИТ-инфраструктуры, предназначенное для мониторинга компьютерных систем и сетей: наблюдения, контроля состояния вычислительных узлов и служб. Обладает более современным и простым в навигации Web-интерфейсом, предлагающим интерактивную информационную панель с обзором хостов, сервисов и сетевых узлов. Система управляется с помощью веб-интерфейса написанного на PHP и использует базу данных MySQL. Недостатком этой системы является большой объем ресурсов, которые занимает база данных. При одном опросе объем данных может достигать 50 мегабайт. Пример интерфейса системы мониторинга Nagios представлен на рисунке 1.7

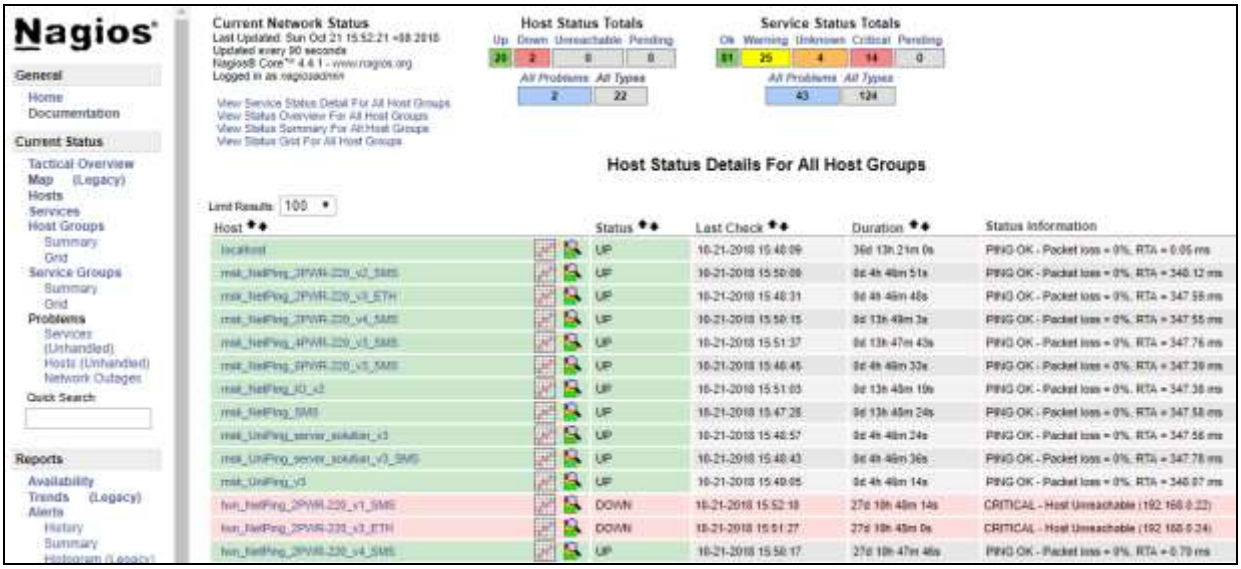

Рисунок 1.7 – Интерфейс системы мониторинга Nagios

Система мониторинга Cacti

Cacti является удобным интерфейсом к RRDTool. С его помощью можно контролировать большое количество различных параметров, таких как загрузку систем и сетей, с выводом всевозможных графиков. Cacti без проблем будет работать в сетях любого размера, как маленьких, так и больших, со сложной разветвленной топологией.

В качестве источника данных могут быть использованы любые внешние команды или сценарии с любыми параметрами, которые нужно собрать, реализована поддержка SNMP. Интерфейс написан на PHP, вся собранная информация сохраняется в базе данных MySQL. Распространяется Cacti по лицензии GNU GPL. Недостатком такой системы является отсутствие возможности проверки состояний датчиков узлов. Можно проверить только его состояние (включен/выключен). Пример интерфейса системы мониторинга Cacti представлен на рисунке 1.8

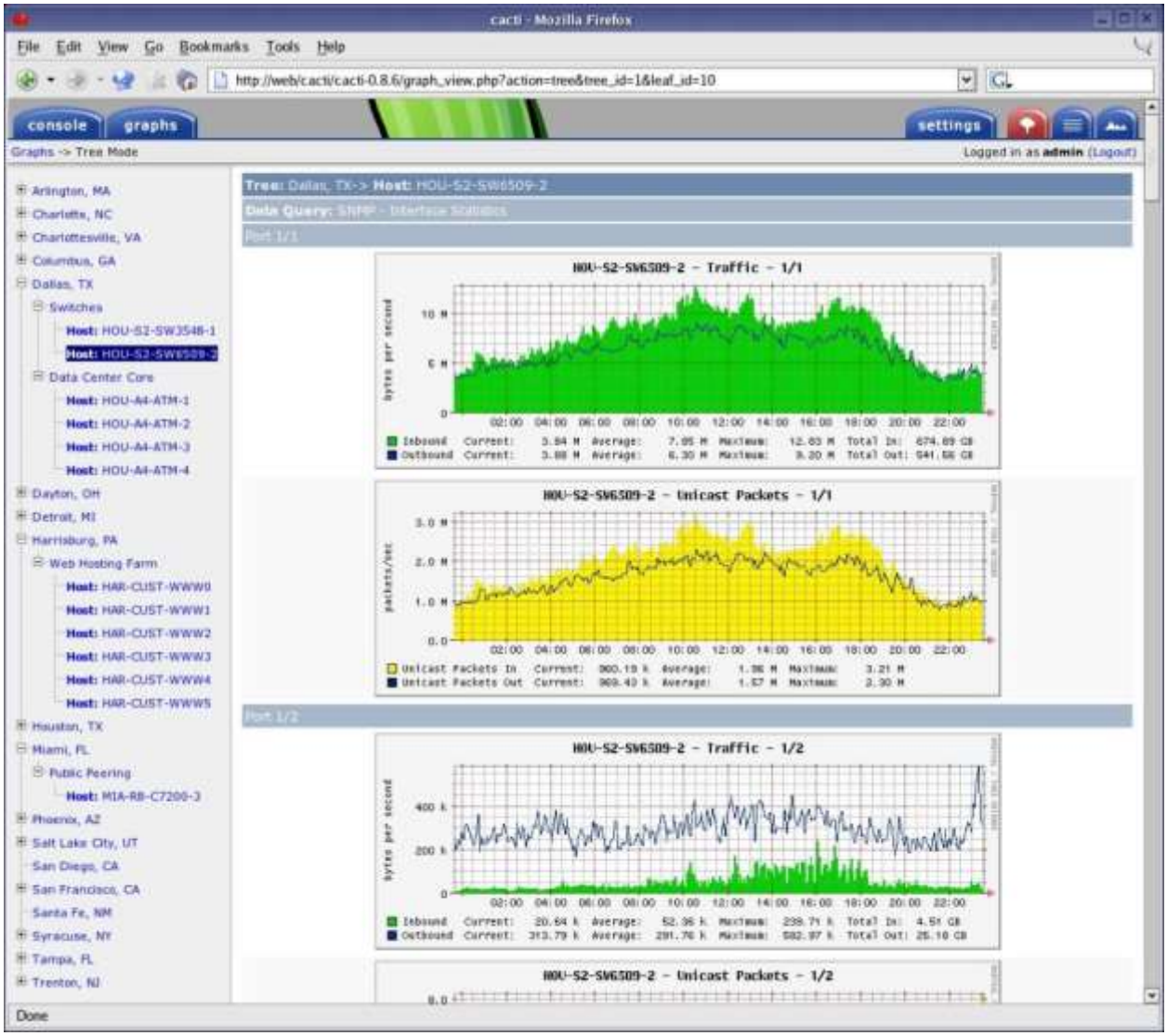

Рисунок 1.8 – Интерфейс системы мониторинга Cacti

Выводы по главе один

Существующие системы мониторинга не позволяют отслеживать состояния большого количества узлов суперкомпьютера и объединять эти данные для общей статистики. На каждом узле опрашивается 8 датчиков, что дает в совокупности 3840 датчиков при 480 узлах. Ни одна система не позволяет просматривать такое большое количество датчиков. даже при 1000 датчиков трафик в сети занимал бы ощутимую часть пропускной способности коммуникационной сети. Также существующие системы мониторинга не способны определять причины выхода из строя узла. Соответственно разработка системы мониторинга аппаратного состояния узлов суперкомпьютера «Торнадо ЮУрГУ» также актуальна.

2 ПРОЕКТИРОВАНИЕ

Данный раздел посвящен проектированию и реализации программной системы для мониторинга аппаратного состояния узлов суперкомпьютера «Торнадо ЮУрГУ».

2.1 Варианты использования системы

Для описания общего представления о функциональном назначении системы была разработана диаграмма вариантов использования. В системе определено одно действующее лицо (актор): администратор суперкомпьютерного центра. Администратор системы имеет следующие варианты использования:

- включить систему мониторинга
- отключить систему мониторинга
- показать статистику по узлам

Диаграмма вариантов использования изображена на рисунке 2.1

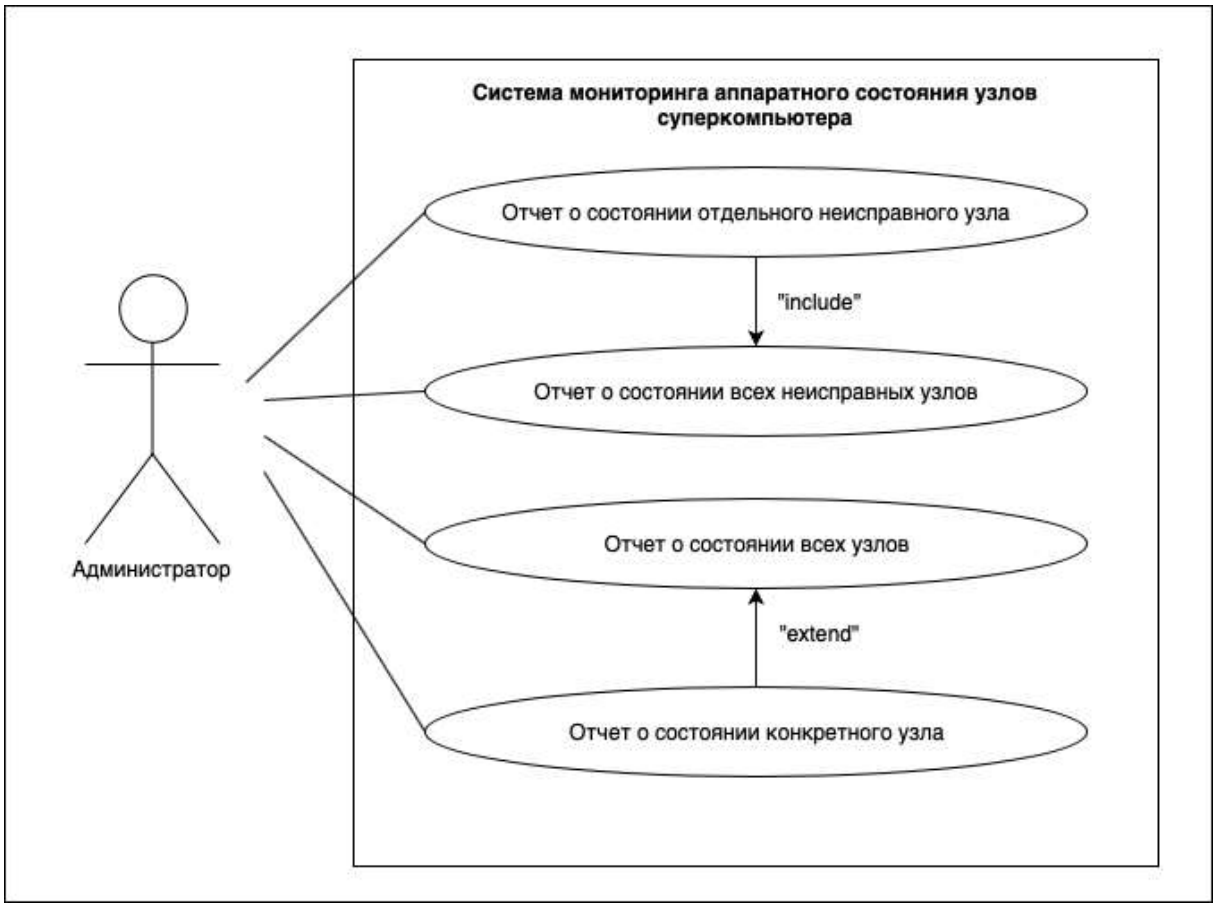

Рисунок 2.1 – Диаграмма вариантов использования

Система мониторинга узлов суперкомпьютера “Торнадо ЮУрГУ” будет осуществлять автоматический сбор и анализ данных с датчиков узлов суперкомпьютера, а также уведомлять пользователя о проблемах и классифицировать найденные ошибки. Полученная информация будет сохраняться в базе данных для возможности последующего использования и отображения пользователю информации об обнаруженной проблеме.

2.2 Функциональные требования к системе

Система должна удовлетворять следующим функциональным требованиям:

1. Оперативно оповещать администратора о появлении проблемных узлов.
2. Отображать информацию о проблеме, ее время и номер узла.
3. Формировать отчет об общем температурном статусе вычислительного кластера.

2.3 Нефункциональные требования к системе

Из-за особенности рабочих условий система мониторинга должна отвечать следующим требованиям:

1. Работа в консольном интерфейсе;
2. Работа в фоновом режиме;
3. Высокая скорость работы, то есть система не должна занимать большое количество ресурсов;
4. Высокая частота опроса датчиков узлов, опрос должен происходить не реже чем раз в минуту;
5. Система должна работать в Linux подобных системах;

2.4 Описание базы данных

Для системы была разработана база данных, в которой используется три таблицы. Таблица Nodes является основной таблицей всей системы, которая является связующей для остальных. Поля и их значения для данной таблицы указаны в таблице 2.1.

Таблица 2.1 – Поля таблицы Node_dim.

Поле таблицы	**Тип**	**Характеристика**
NodeKey	Численный	Уникальный ключ связи таблицы
Node_ID	Численный	Уникальный номер узла
Node_Name	Текстовый	Название узла

В Таблице Sensors хранятся показания с датчиков на вычислительном узле. Поля и из значения для данной таблицы указаны в таблице 2.2.

Таблица 2.2 – Поля таблицы Sensors.

Поле таблицы	**Тип**	**Характеристика**
SensorKey	Численный	Уникальный ключ связи таблицы
Temp	Численный	Температура центрального процессора

Продолжение таблицы 2.2

12V	Численный	Напряжение по линии питания 12 вольт
5V	Численный	Напряжение по линии питания 5 вольт
BatV	Численный	Напряжение элемента питания

В таблице Rack хранится принадлежность узла определенной стойке суперкомпьютера. Поля и их значения для данной таблицы указаны в таблице 2.3.

Таблица 2.3 – Поля таблицы Racks.

Поле таблицы	**Тип**	**Характеристика**
Rack_ID	Численный	Порядковый идентификатор в таблице
Node_Name	Текстовый	Номер стойки суперкомпьютера

В таблице Time_dim хранится время получения показаний с датчиков вычислительного узла. Поля и их значения для данной таблицы указаны в таблице 2.4.

Таблица 2.4 – Поля таблицы Time_dim.

Поле таблицы	**Тип**	**Характеристика**
TimeKey	Численный	Уникальный ключ связи таблицы
Day	Дата	День месяца
Day_of_week	Дата	День недели
Month	Дата	Месяц
Week	Дата	Неделя

В таблице Sensor_Fact хранятся ключи связи с другими таблицами. Поля и их значения для данной таблицы указаны в таблице 2.5.

Таблица 2.5 – Поля таблицы Node_dim.

Поле таблицы	**Тип**	**Характеристика**
TimeKey	Численный	Уникальный ключ связи таблицы
NodeKey	Численный	Уникальный ключ связи таблицы
SensorKey	Численный	Уникальный ключ связи таблицы

На рисунке 2.2 изображена схема базы данных, которая была разработана по общему строению самой базы данных.

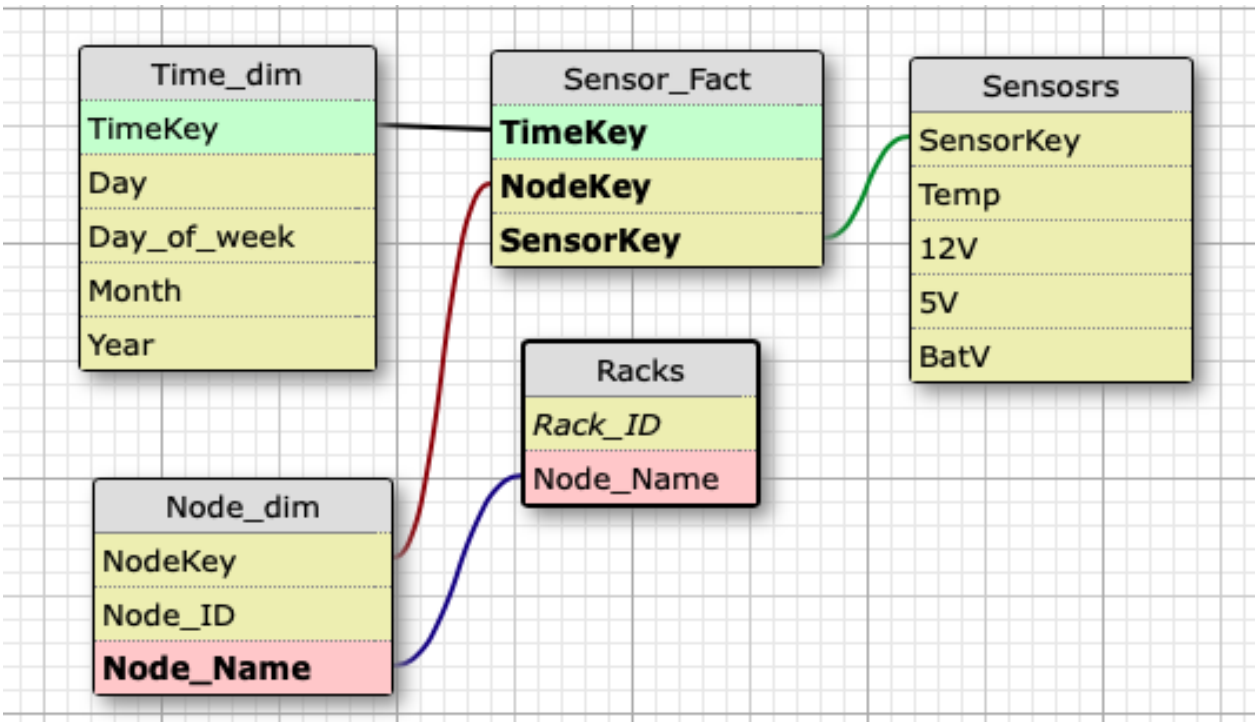

Рисунок 2.2 – Схема базы данных

Для сохранения целостности базы данных в системе были определены ограничения, представленные в таблице 2.4

Таблица 2.4 – Ограничения данных

Имя таблицы	Имя поля	Ограничение
Node_dim	NodeKey	уникальность
Node_dim	Node_Name	уникальность
Sensors	SensorKey	уникальность
Racks	Rack_ID	уникальность
Sensors	Temp	больше нуля
Time_dim	TimeKey	уникальность

Выводы по главе два

Существующие решения требуют установки дополнительных модулей на вычислительные узлы. Фоновые процессы будут занимать конечную производительность вычислительных узлов. В готовых системах мониторинга объем базы данных имеют неоптимизированную структуру и занимают большой объем дискового пространства. Для этого была спроектированная собственная база данных, которая учитывает особенности проектирования суперкомпьютера.

3 РЕАЛИЗАЦИЯ

3.1 Используемые технологии

Для реализации предложенной системы использовался язык программирования Python 3.6 [13];

Для кодирования системы использовалась интегрированная среда разработки PyCharm Community Edition 2017.3.3 [14]

Для разработки системы мониторинга будут использоваться следующие библиотеки:

- Библиотека SQLite3 [15];
- Библиотека Threading [16];
- Библиотека Subprocess [17];
- Библиотека Re [18];

Библиотека SQLite3

Это постоянно совершенствуемая библиотека, осуществляющая работу автономного механизма баз данных SQL, который не нуждается в сервере. Код, используемый в SQLite является открытым, что позволяет использовать данную библиотеку в любых целя – коммерческих или частных. Преимуществом SQLite3 в разработке системы мониторинга является значительная автономность. Она требует минимальной поддержки от внешних библиотек или от операционной системы для работы с базой данных.

Еще один аргумент в пользу SQLite — ее компактность. Со всеми включенными настройками, в зависимости от параметров настройки оптимизации компилятора, размер библиотеки может составлять менее 250 Кб. Если опущены дополнительные настройки, размер библиотеки SQLite может быть сокращен до 180 Кб и немного меньше. SQLite может работать при минимальном стеке около 16 Кб и очень небольшой динамической памяти — 100 Кб. Это позволяет использовать SQLite как механизм базы данных в системе мониторинга аппаратного состояния узлов суперкомпьютера.

В данном случае важно соотношение между использованием памяти и скоростью. Чем больше отводится памяти для SQLite, тем быстрее она работает. Тем не менее рабочие характеристики обычно и так достаточно высоки даже в средах с низким объемом памяти. Объем всей базы данных для 480 узлов составляет всего 459 Килобайт, что значительно меньше, чем у сторонних систем мониторинга.

Программная библиотека Threading

Библиотека threading была введена в язык Python с версии 1.5.2 и совершенствуется по сей день. Задача данной библиотеки значительно упростить работу с потоками и позволить программировать запуск нескольких операций одновременно. В данном случае, опрос узлов будет производиться с помощью этой библиотеки в 8 потоков. По сравнению с последовательным опросом, использование данной библиотеки сокращает время опроса всех 480 узлов до 42 секунд, по сравнению с 310 секундами. Принцип использования библиотеки threading при опросе узлов изображен на рисунке 3.1

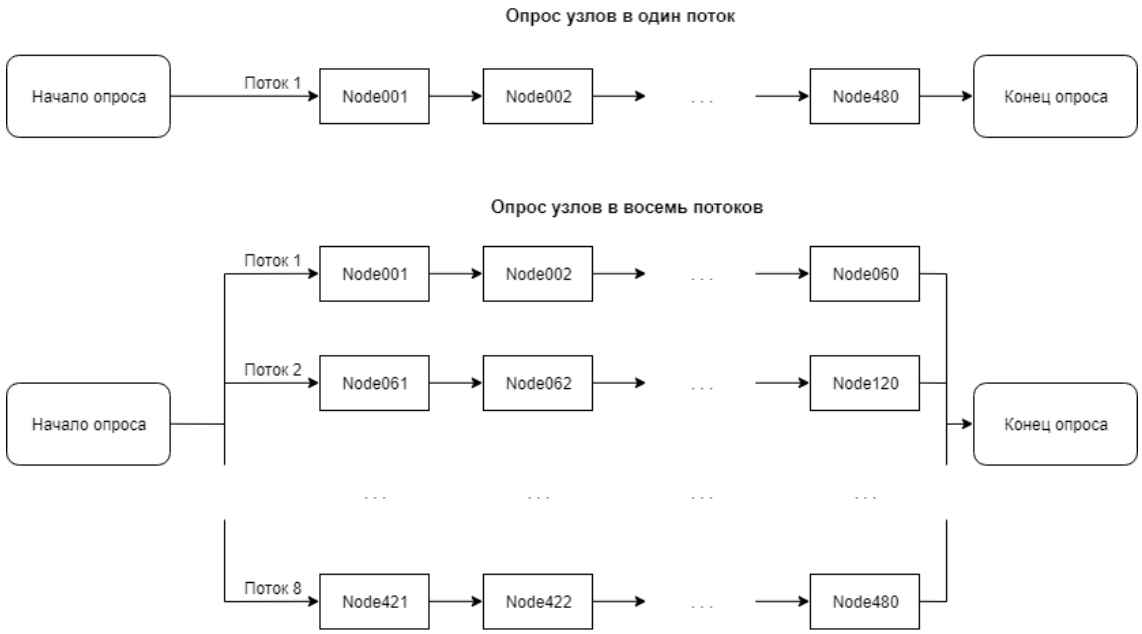

Рисунок 3.1 – Опрос узлов суперкомпьютера

Программная библиотека Subprocess

Модуль Subprocess обеспечивает унифицированный интерфейса доступа к операционной системе. Модуль Subprocess отвечает в языке Python за выполнение следующих действий: порождение новых процессов, соединение с потоками стандартного ввода, стандартного вывода, стандартного вывода сообщений об ошибках и получение кодов возврата от этих процессов.

Для запуска команд системы используется две функции subprocess.call() и subprocess.Popen(). Основное отличие этих функций между собой: функция subprocess.call() блокирует выполнение сценария до получения ответа, в то время как функция subprocess.Popen() – нет.

subprocess.call() – используется, если нужно просто запустить команду, а вывод от нее сохранять не требуется.

subprocess.Popen() – используется, если требуется захватить вывод команды

При разработке системы мониторинга будет использоваться функция subpocess.Popen() т.к. нам необходимо получить все показания датчиков с узла суперкомпьютера и лишь затем записывать их в базу данных.

Программная библиотека Re

Библиотека для работы с регулярными выражениями. Регулярные выражение – это последовательность символов, используемая для поиска и замены текста в строке или в файле. В последние годы языки общего назначения стали чаще использоваться для анализа данных. Разработчики используют Python для решения своих задач. И в этом помогают регулярные выражения. Регулярные выражения – инструмент для упорядочивания, поиска или извлечения текстовых данных. В данном случае эта библиотека необходима для извлечения показаний датчиков.

3.2 Основные методы работы системы мониторинга

Алгоритм опроса узлов суперкомпьютера

В момент запуска система проверяет наличие базы данных, при ее отсутствии система автоматически при помощи библиотеки SQLite3 создаст базу данных. Система производит опрос датчиков узлов суперкомпьютера раз две минуты, создавая 8 параллельных потоков и записывает все показания в базу данных. Для каждого узла создается три поля, с его порядковым номером, показаниями датчиков и временем записи показаний.

На рисунке 3.2 представлен запуск опроса узлов суперкомпьютера в несколько потоков с помощью метода Thread, во входных параметрах указывается диапазон узлов для опроса.

```
thread1 = Thread(target=node_statistic, args=(1,60))
thread2 = Thread(target=node_statistic, args=(61,120))
thread3 = Thread(target=node_statistic, args=(121,180))
thread4 = Thread(target=node_statistic, args=(181,240))
thread5 = Thread(target=node_statistic, args=(241,300))
thread6 = Thread(target=node_statistic, args=(301,360))
thread7 = Thread(target=node_statistic, args=(361,420))
thread8 = Thread(target=node_statistic, args=(421,480))
```

Рисунок 3.2 – Запуск опроса узлов в 8 потоков

В приложении А представлен метод *create_db*, который создает файл базы данных *nodes_statistik.db*, куда будут сохранятся показания датчиков узлов суперкомпьютера.

На рисунке 3.3 представлен метод *add_to_db*, который записывает показания датчиков узла в базу данных.

```
def add_to_db(massiv):
    with sqlite3.connect("nodes_database.db") as con:
        cur= con.cursor()
        cur.execute("INSET INTO Sensor_Fact VALUES(?, ?, ?)", (massiv))
        con.commit()
```

Рисунок 3.3 – Метод добавление данных в базу

Системе необходимо обработать входные данные с датчиков узла, для этого вызывается отдельный метод *node_statistic*, который получает данные с узла, формирует их и записывает в локальную базу данных. Метод представлен на рисунке 3.4

```
def node_statistic(innode, outnode):
    for i in range(innode, outnode+1):
        number = str(i)
        if (i < 10):
            nodenumber = "node00" + number
        elif (i < 100):
            nodenumber = "node0" + number
        else:
            nodenumber = "node" + number

        bashRvit = "rvitals " + nodenumber
        RvitRes = run_bash(bashRvit)
        if (RvitRes==''):
      update_in_db(str(nodenumber)+' OFF', nodenumber)
   else:
      update_in_db(RvitRes, nodenumber)
```

Рисунок 3.4 – Метод добавление данных в базу

Методы вывода отчетов

Администратор суперкомпьютера может запросить отчет об проблемных узлах суперкомпьютера, с помощью ключа запуска *warningnodes*.

С помощью запросов к базе данных, система получает показания с каждого узла и анализирует их. Если показания выходят за рамки допустимых, то система автоматически добавляет узел в массив проблемных узлов и помечает причину проблемы: узел отключен, повышенная температура, низкое напряжение батарейки CMOS, низкое или высокое напряжение по линии 5 или 12 вольт. После проверки всех узлов система выводит в консоль список всех проблемных узлов.

На рисунке 3.5 представлен метод *get_from_db*, который берет показания датчиков узла из базы данных.

```
def get_from_db(nodenumber):
    with sqlite3.connect("nodes_database.db") as con:
        cur = con.cursor()
        cur.execute("SELECT sensors FROM Sensors WHERE title=?", [(node-
number)])
        return(cur.fetchall())
```

Рисунок 3.5 – Метод чтения из базы данных

На рисунке 3.6 представлен отчет о состоянии проблемных узлов, где в каждой строке указывается имя узла и соответствующую ошибку.

```
node040 bad temperature
node157  OFF
node175  OFF
node180  bad temperature
node241  OFF
node242  OFF
node243  bad CMOS
node244  OFF
node245  OFF
node246  OFF
node247  bad temperature
node248  OFF
node249  bad temperature
node250  OFF
node351  bad 12 voltage
```

Рисунок 3.6 – Отчет системы о состоянии проблемных узлов

Администратор с помощью ключа запуска oknodes может получить отчет об всех исправных узлах суперкомпьютера. На рисунке 3.7 представлен отчет о состоянии исправных узлов, где в каждой строке выводятся исправный узел.

```
node121 OK!
node122 OK!
node123 OK!
node125 OK!
node126 OK!
node127 OK!
node128 OK!
node133 OK!
node134 OK!
node135 OK!
node136 OK!
node137 OK!
node138 OK!
node139 OK!
node140 OK!
node144 OK!
node145 OK!
```

Рисунок 3.7 – Отчет системы о состоянии исправных узлов

Администратор может получить отчет о состоянии конкретного узла, для этого в ключе запуска указывается имя данного узла, к примеру *node260*. На рисунке 3.8 представлен отчет о состоянии конкретного узла.

```
node260: +1.5V: 1.504 Volts
node260: +12V: 12 Volts
node260: +3.3V: 3.192 Volts
node260: +3.3VSB: 3.336 Volts
node260: +5V: 5.08 Volts
node260: CPU1 Temp: 0
node260: CPU1 Vcore: 1.192 Volts
node260: CPU1DIMM: 1.528 Volts
node260: CPU2 Temp: 0
node260: CPU2 Vcore: 1.24 Volts
node260: CPU2DIMM: 1.52 Volts
node260: Fan1: N/A
node260: Fan2: N/A
node260: Fan3: N/A
node260: Fan4: N/A
node260: P1-DIMM1A Temp: 36 C (97 F)
node260: P1-DIMM1B Temp: N/A
node260: P1-DIMM2A Temp: 37 C (99 F)
node260: P1-DIMM2B Temp: N/A
node260: P1-DIMM3A Temp: 37 C (99 F)
node260: P1-DIMM3B Temp: N/A
node260: P2-DIMM1A Temp: 35 C (95 F)
node260: P2-DIMM1B Temp: N/A
node260: P2-DIMM2A Temp: 31 C (88 F)
node260: P2-DIMM2B Temp: N/A
node260: P2-DIMM3A Temp: 30 C (86 F)
node260: P2-DIMM3B Temp: N/A
node260: PS Status: 0
node260: System Temp: 44 C (111 F)
node260: VBAT: 3.336 Volts
```

Рисунок 3.8 – Отчет системы о состоянии узла

3.3 Тестирование системы

Важным требованием к работе системы является скорость опроса аппаратной части узлов суперкомпьютера. Для этого проведем тестирование времени опроса разного количества узлов суперкомпьютера.

На рисунке 3.9 представлен результат опроса 480 узлов суперкомпьютера

```
Timer start

thread from 1 to 60 started
thread from 61 to 120 started
thread from 121 to 180 started
thread from 181 to 240 started
thread from 241 to 300 started
thread from 301 to 360 started
thread from 361 to 420 started
thread from 421 to 480 started
thread from 1 to 60 join
thread from 61 to 120 join
thread from 121 to 180 join
thread from 181 to 240 join
thread from 241 to 300 join
thread from 301 to 360 join
thread from 361 to 420 join
thread from 421 to 480 join

Done! Time: 37.74 Sec
```

Рисунок 3.9 – Опрос 480 узлов

На рисунке 3.10 представлен результат опроса 420 узлов суперкомпьютера

```
Timer start

thread from 1 to 60 started
thread from 61 to 120 started
thread from 121 to 180 started
thread from 181 to 240 started
thread from 241 to 300 started
thread from 301 to 360 started
thread from 361 to 420 started
thread from 1 to 60 join
thread from 61 to 120 join
thread from 121 to 180 join
thread from 181 to 240 join
thread from 241 to 300 join
thread from 301 to 360 join
thread from 361 to 420 join

Done! Time: 35.21 Sec
```

Рисунок 3.10 – Опрос 420 узлов

На рисунке 3.11 представлен результат опроса 360 узлов суперкомпьютера

```
Timer start

thread from 1 to 60 started
thread from 61 to 120 started
thread from 121 to 180 started
thread from 181 to 240 started
thread from 241 to 300 started
thread from 301 to 360 started
thread from 1 to 60 join
thread from 61 to 120 join
thread from 121 to 180 join
thread from 181 to 240 join
thread from 241 to 300 join
thread from 301 to 360 join

Done! Time: 36.48 Sec
```

Рисунок 3.11 – Опрос 360 узлов

На рисунке 3.12 представлен результат опроса 300 узлов суперкомпьютера

```
Timer start

thread from 1 to 60 started
thread from 61 to 120 started
thread from 121 to 180 started
thread from 181 to 240 started
thread from 241 to 300 started
thread from 1 to 60 join
thread from 61 to 120 join
thread from 121 to 180 join
thread from 181 to 240 join
thread from 241 to 300 join

Done! Time: 31.10 Sec
```

Рисунок 3.12 – Опрос 360 узлов

На рисунке 3.13 представлен результат опроса 240 узлов суперкомпьютера

```
Timer start

thread from 1 to 60 started
thread from 61 to 120 started
thread from 121 to 180 started
thread from 181 to 240 started
thread from 1 to 60 join
thread from 61 to 120 join
thread from 121 to 180 join
thread from 181 to 240 join

Done! Time: 27.81 Sec
```

Рисунок 3.13 – Опрос 240 узлов

На рисунке 3.14 представлен результат опроса 180 узлов суперкомпьютера

```
Timer start

thread from 1 to 60 started
thread from 61 to 120 started
thread from 121 to 180 started
thread from 1 to 60 join
thread from 61 to 120 join
thread from 121 to 180 join

Done! Time: 31.50 Sec
```

Рисунок 3.14 – Опрос 180 узлов

На рисунке 3.15 представлен результат опроса 120 узлов суперкомпьютера

```
Timer start

thread from 1 to 60 started
thread from 61 to 120 started
thread from 1 to 60 join
thread from 61 to 120 join

Done! Time: 27.68 Sec
```

Рисунок 3.15 – Опрос 120 узлов

На рисунке 3.16 представлен результат опроса 60узлов суперкомпьютера

```
Timer start

thread from 1 to 60 started
thread from 1 to 60 join

Done! Time: 24.10 Sec
```

Рисунок 3.16 – Опрос 60 узлов

Для сравнения произведем последовательный опрос 480 узлов в один поток.

На рисунке 3.17 представлен результат последовательного опроса 480 узлов суперкомпьютера

```
Timer start

thread from 1 to 480 started
thread from 1 to 480 join

Done! Time: 317.10 Sec
```

Рисунок 3.17 – Последовательный опрос 480 узлов

Получившиеся результаты были сведены в диаграмму. Диаграмма зависимости времени опроса от количества опрашиваемых узлов приведена на рисунке 3.18

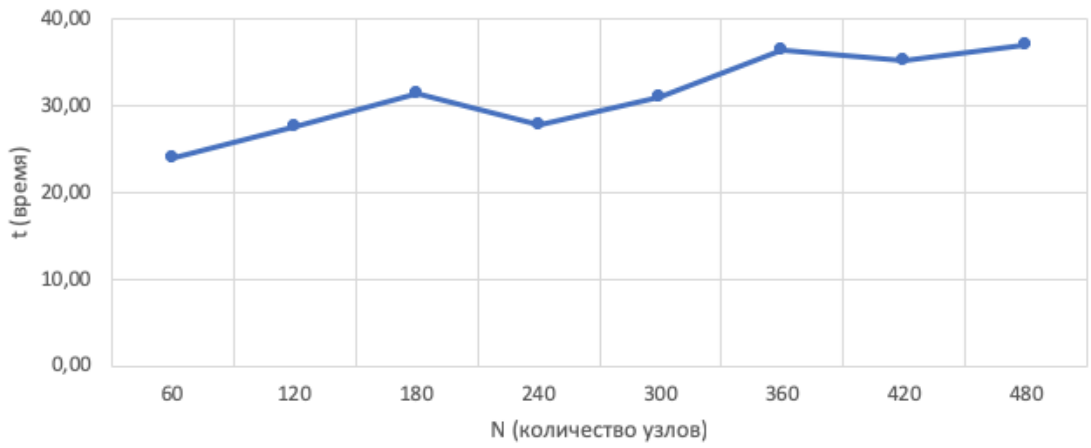

Рисунок 3.18 – Диаграмма зависимости

По результатам тестирования на рисунке 3.18 видно, что среднее время опроса всех 480 узлов в многопоточном режиме суперкомпьютера составляет 35 секунд, в отличии времени однопоточного режима, которое составляет 317 секунд.

Вывод по главе три

Разработанная система мониторинга аппаратной части узлов, соответствует всем поставленным требованиям к разработке программной системы. Система имеет высокую скорость опроса узлов, которая составляет 35 секунд. Все методы формирования отчетов имеют высокую информативность для администратора лаборатории суперкомпьютерного моделирования.

ЗАКЛЮЧЕНИЕ

1. Произведен анализ распределенного и централизованного типа архитектуры систем мониторинга. Для разработки системы мониторинга аппаратной части узлов суперкомпьютера «Торнадо ЮУрГУ» был выбран централизованный тип архитектуры.

2. После обзора существующих систем мониторинга аппаратного состояния узлов было принято решение разработать собственную систему мониторинга.

3. На языке Python с использованием сторонних библиотек было разработано консольное приложение системы мониторинга аппаратной части узлов суперкомпьютера торнадо, которое занимает 2 Мб оперативной памяти и использует лишь 2 % общей производительности узла в пиковой нагрузке (момента опроса узлов)

4. В системе с помощью методов был реализован требуемый функционал консольного приложения, а именно вывод отчетов по узлам, вывод статистики по узлам и вывод показаний отдельных узлов.

5. Полученная система имеет высокую гибкость при применении и может быть расширена для снятий показаний суперкомпьютеров с количеством вычислительных узлов до двух тысяч.

В результате выполнения работы были решены все поставленные задачи, таким образом, цель данной работы достигнута.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 TOP500 Supercomputer Sites [Электронный ресурс]. / E. Strohmaier, J. Dongarra, H. Simon, M. Meuer. – Режим доступа: http://top500.org/ – Заглавие с экрана.

2 Zabbix [Электронный ресурс]. – Режим доступа: http://www.zabbix.com – Заглавие с экрана.

3 Nagios – система мониторинга [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/307832/ – Заглавие с экрана.

4 Supermicro IPMI utilities [Электронный ресурс]. – Режим доступа: https://www.supermicro.com/en/solutions/management-software/ipmi-utilities – Заглавие с экрана.

5 Хомутский, Ю.В. Журнал сетевых решений/LAN // Современные суперкомпьютеры. -2013. No.5. –С. 8-17.

6 Zenoss Community – Open Source Network Monitoring and Systems Management [Электронный ресурс]. Режим доступа: http://www.zenoss.org – Заглавие с экрана.

7 Cacti® – The Complete RRDTool-based Graphing Solution [Электронный ресурс]. – Режим доступа: http://www.cacti.net – Заглавие с экрана.

8 Massie, M.L. The ganglia distributed monitoring sys-tem: design, implementation, and experience / M.L. Massie, B.N. Chun, D.E. Culler. – Parallel Computing. 2004. – Vol. 30, No. 7. – P. 817–840.

9 The OpenNMS Project [Электронный ресурс]. – Режим доступа: http://www.opennms.org – Заглавие с экрана.

10 Nagios – The Industry Standard in IT Infrastructure Monitoring [Электронный ресурс]. – Режим доступа: http://www.nagios.org – Заглавие с экрана.

11 Collectd – The system statistics collection daemon [Электронный ресурс]. – Режим доступа: https://collectd.org – Заглавие с экрана.

12 Gunter D. [et al.]. Dynamic monitoring of high-performance distributed applications // Proceedings 11th IEEE International Symposium on High Performance Distributed Computing: IEEE Comput. Soc. – 2002. – P. 163–170.

13 Основы языка программирования Python [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/31180/ – Заглавие с экрана.

14 PyCharm – интеллектуальная Python IDE [Электронный ресурс]. – Режим доступа: https://jetbrains.ru/products/pycharm/ – Заглавие с экрана.

15 Руководство по SQLite [Электронный ресурс]. – Режим доступа: https://proglib.io/p/sqlite-tutorial/ – Заглавие с экрана

16 Модуль Threading на примерах [Электронный ресурс]. – Режим доступа: http://python-3.ru/page/import-threading – Заглавие с экрана.

17 Модуль subprocess – Работаем с процессами [Электронный ресурс]. – Режим доступа: https://python-scripts.com/subprocess – Заглавие с экрана.

18 Использование регулярных выражений в Python [Электронный ресурс]. – Режим доступа: https://tproger.ru/translations/regular-expression-python/ – Заглавие с экрана.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А. ЛИСТИНГСОЗДАНИЯ БАЗЫ ДАННЫХ

```
-- ---
-- Globals
-- ---

-- SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
-- SET FOREIGN_KEY_CHECKS=0;

-- ---
-- Table 'Sensor_Fact'
--
-- ---

DROP TABLE IF EXISTS `Sensor_Fact`;

CREATE TABLE `Sensor_Fact` (
  `TimeKey` DATETIME NULL DEFAULT NULL,
  `NodeKey` INT NULL DEFAULT NULL,
  `SensorKey` INTEGER NULL DEFAULT NULL,
  PRIMARY KEY (`TimeKey`, `NodeKey`, `SensorKey`)
);

-- ---
-- Table 'Time_dim'
--
-- ---

DROP TABLE IF EXISTS `Time_dim`;

CREATE TABLE `Time_dim` (
  `TimeKey` DATETIME NULL DEFAULT NULL,
  `Day` INTEGER NULL DEFAULT NULL,
  `Day_of_week` INTEGER NULL DEFAULT NULL,
  `Month` INTEGER NULL DEFAULT NULL,
  `Year` INTEGER NULL DEFAULT NULL,
  PRIMARY KEY ()
);

-- ---
-- Table 'Sensosrs'
--
-- ---

DROP TABLE IF EXISTS `Sensosrs`;

CREATE TABLE `Sensosrs` (
  `SensorKey` INTEGER NULL DEFAULT NULL,
  `Temp` INTEGER NULL DEFAULT NULL,
```

```
  `12V` INTEGER NULL DEFAULT NULL,
  `5V` INTEGER NULL DEFAULT NULL,
  `BatV` INTEGER NULL DEFAULT NULL,
  PRIMARY KEY ()
);

-- ---
-- Table 'Node_dim'
--
-- ---

DROP TABLE IF EXISTS `Node_dim`;

CREATE TABLE `Node_dim` (
  `NodeKey` INTEGER NULL DEFAULT NULL,
  `Node_ID` INTEGER NULL DEFAULT NULL,
  `Node_Name` MEDIUMTEXT NULL DEFAULT NULL,
  PRIMARY KEY (`Node_Name`)
);

-- ---
-- Table 'Racks'
--
-- ---

DROP TABLE IF EXISTS `Racks`;

CREATE TABLE `Racks` (
  `Rack_ID` INTEGER NULL AUTO_INCREMENT DEFAULT NULL,
  `Node_Name` MEDIUMTEXT NULL DEFAULT NULL,
KEY (`Rack_ID`)
);

-- ---
-- Foreign Keys
-- ---

ALTER TABLE `Time_dim` ADD FOREIGN KEY (TimeKey) REFERENCES `Sensor_Fact`
(`TimeKey`);
ALTER TABLE `Sensosrs` ADD FOREIGN KEY (SensorKey) REFERENCES `Sensor_Fact`
(`SensorKey`);
ALTER TABLE `Node_dim` ADD FOREIGN KEY (NodeKey) REFERENCES `Sensor_Fact`
(`NodeKey`);
ALTER TABLE `Racks` ADD FOREIGN KEY (Node_Name) REFERENCES `Node_dim`
(`Node_Name`);

-- ---
-- Table Properties
-- ---
```

```
-- ALTER TABLE `Sensor_Fact` ENGINE=InnoDB DEFAULT CHARSET=utf8 COL-
LATE=utf8_bin;
-- ALTER TABLE `Time_dim` ENGINE=InnoDB DEFAULT CHARSET=utf8 COL-
LATE=utf8_bin;
-- ALTER TABLE `Sensosrs` ENGINE=InnoDB DEFAULT CHARSET=utf8 COL-
LATE=utf8_bin;
-- ALTER TABLE `Node_dim` ENGINE=InnoDB DEFAULT CHARSET=utf8 COL-
LATE=utf8_bin;
-- ALTER TABLE `Racks` ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- ---
-- Test Data
-- ---

-- INSERT INTO `Sensor_Fact` (`TimeKey`,`NodeKey`,`SensorKey`) VALUES
-- ('','','');
-- INSERT INTO `Time_dim` (`TimeKey`,`Day`,`Day_of_week`,`Month`,`Year`) VALUES
-- ('','','','','');
-- INSERT INTO `Sensosrs` (`SensorKey`,`Temp`,`12V`,`5V`,`BatV`) VALUES
-- ('','','','','');
-- INSERT INTO `Node_dim` (`NodeKey`,`Node_ID`,`Node_Name`) VALUES
-- ('','','');
-- INSERT INTO `Racks` (`Rack_ID`,`Node_Name`) VALUES
-- ('','');
```