*УДК ?*

Андресюк Богдан

Бойкив Тарас

Паламарчук Валерий

Щербина Александр

Национальный технический университет Украины «КПИ», Киев, Украина

**Обеспечение уровней безопасности и качества обслуживания в Private Cloud системе реального времени**

**Аннотация.** Предложено улучшение алгоритма планирования А\* для использования в Private Cloud системах реального времени. Алгоритм в связке с алгоритмом разделения секрета Shamir позволяет обеспечить статическое планирование с учетом уровня безопасности и требуемого времени отклика для задач. Обеспечивается безопасность доставки задач в Cloud систему и результатов обратно к пользователю.

**Анотація.** Запропоновано покращення алгоритму планування А\* для використання в Private Cloud системах реального часу. Алгоритм у парі з алгоритмом розподілу даних Shamir дозволяє забезпечити статичне планування з урахуванням рівня безпеки і вимагаємого часу відгуку системи для задач. Забезпечується безпека доставки задач до Cloud системи і результатів назад до користувача.

**Annotation.** Proposed improvement of the scheduling algorithm A \* for use in the Private Cloud real-time systems. Algorithm in composition with the Shamir secret sharing algorithm gives: static task scheduling, high level security and the required response times for the tasks. Ensures safe delivery tasks in Cloud system and the results back to the user.

**Ключевые слова**. Private Cloud, система реального времени, планирование распределения задач.

**Ключові слова**. Private Cloud, система реального часу, планування розподілу задач.

**Keywords.** Private Cloud, real time system, task scheduling.

**Постановка проблемы.** Запуск задач с требованиями реального времени в Cloud системе, накладывает определенный набор требований к уровню обслуживания задач в Cloud системе. Использование стандартных алгоритмов планирования и обеспечения безопасности в Cloud системе не позволяет удовлетворять эти требования.

**Анализ последних исследований и публикаций.** Согласно последним данным NIST (Национальный институт стандартов и технологий США) "Cloud Computing Synopis and Recommendations"[1] частное облако – облачная инфраструктура, которая эксплуатируется исключительно организацией, может находиться под управлением организации или третьей стороны, и быть собственностью организации или третьей стороны.

**Периметр безопасности**

**Периметр безопасности**

\*\*\*

Клиент

Клиент

\*\*\*

Клиент

Клиент

А

В

Рис. 1 А- Собственное частное облако, В - стороннее частное облако

При использовании собственного частного облака нет необходимости полагаться на внешние сети, а также легче обеспечить внутреннюю безопасность. Но для создания и поддержки этой инфраструктуры потребуются значительно большие средства, чем при использовании сторонних сервисов. Кроме того, доступно меньшее количество ресурсов для расширения облака при внезапной нарастающей вычислительной нагрузке. Таким образом теряются некоторые из ключевых преимуществ облачных систем.

При использовании стороннего частного облака, появляется большая гибкость в предоставлении дополнительных ресурсов в короткие сроки. Но данная инфраструктура будет частично зависеть от внешних сетей, что потребует усилий для обеспечения безопасности.

В частном облаке все ресурсы объединены в пулы, что позволяет достичь высокой эффективности их использования и масштабируемости при выделении ресурсов под определенные задачи. Распределяя ресурсы из общего пула между несколькими задачами и подразделениями, можно повысить эффективную загрузку имеющихся ресурсов.

После объединения ресурсов в пулы появляется возможность в автоматизированном порядке увеличивать и уменьшать объем ресурсов, выделенных под конкретную задачу. Фактически, это позволяет быстро масштабировать сервисы в соответствии с требованиями клиента.

При запросе, настройке и управлении поставщики услуг и потребители используют интерактивный портал или систему, предназначенную для автоматического предоставления ресурсов.

Для того чтобы задачи с требованиями реального времени могли выполнятся в Cloud системе, она должна удовлетворять следующим требованиям.

**Основные требования для облачных систем реального времени**

1. Высокоскоростные источники данных. Эти данные должны быть вытеснены в облако, а затем к пользователю. Опрос готовности данных требует слишком много времени и потребляет слишком много пропускной способности.[2]

2. Низкие задержки передачи данных. Данные должны передаваться быстро и легко с помощью системы в базу данных реального времени. Реляционные базы данных, обычно используемые для бизнес-систем, являются слишком медленными.

3. Несколько типов пользователей. Разные пользователи, такие как веб-браузеры, базы данных, электронные таблицы должны иметь доступ к одному источнику данных.[3]

4. Независимая система резервирования. Необходимо предоставить возможность обеспечить полное резервирование каналов передачи данных, которые будут задействованы в случае любого перерыва в обслуживании.

5. Локальная синхронизация. Система должна содержать полную копию данных с источника, и в реальном времени отправлять их пользователям по локальной сети. Если канал связи с облаком прерван, локальные клиенты и серверы не должны реагировать на сбой в работе сети. Вместо этого должны появиться отдельные области управления, которые смогут продолжить работу как "островки автоматизации", пока связь с облаком не будет восстановлена.[4]

**Основные нерешенные части общей проблемы.** Удовлетворение одновременно требований систем реального времени и Cloud системы сложная задача. Необходимо обеспечить заданное время отклика задачи и обеспечить гибкость выделения ресурсов в Cloud системе. Основными проблемами являются: выполнение планирования с учетом обеспечения заданного времени отклика, обеспечения требуемого уровня безопасности.

**Цель статьи.** Предложены улучшения существующего алгоритма для выполнения планирования в Cloud системе с учетом требований задач реального времени, а также предложена схема обеспечения безопасности как внутри Cloud системы так и при пересылки задач от клиента и при отправке результатов клиенту.

**Изложение основного материала статьи. Описание модели СРВ**

N

**Cloud система реального времени**

N

N – канальное соединение

N

**Группа клиентов**

**Уровень безопасности N**

**Группа клиентов**

**Уровень безопасности 1**

Входная очередь

Выходная очередь

Клиент

Клиент

SSA

Канал обслуживания

Планировщик

Канал обслуживания

\*\*\*

Клиент

Клиент

SSA

SSA

SSA

N

SSA

SSA

N

\*\*\*

Рис. 2 Структура разработанной модели Cloud системы реального времени

Клиенты системы разделяются на группы по уровню безопасности которую они требуют от системы и приоритетов. Для клиентов которые требуют уровень безопасности выше минимального, является необходимым иметь несколько каналов связи с Cloud системой, для возможности передачи частей задачи по разным каналам. Клиентам с более высоким уровнем приоритета гарантируется большая вероятность безотказной работы.

Для клиентов с уровнем безопасности выше минимального, задача разделяется по алгоритму Шамира на n частей и по разным каналам доставляется во входную очередь системы где и собирается из частей.

Если же пользователь требует только минимальный уровень безопасности, то задача доставляется без деления по одному каналу связи.

Во входной очереди накапливаются задачи на выполнение. Каждая задача имеет такие параметры: приоритет, уровень безопасности, требования к оборудованию (объем памяти, частота процессора), время выполнения на требуемом оборудовании, время за которое должен быть получен результат.

Через заданный в системе промежуток времени или после накопления определенного количества задач во входной очереди, производится планирование распределения задач по ресурсам.

Каждый ресурс системы имеет набор параметров: объем памяти, количество ядер и частоту процессора, уровень безопасности – определяется степенью защищенности машины.

После планирования задачи доставляются на каналы обслуживания, где происходит выполнение заданий.

После завершения обработки результаты задач попадают в выходную очередь.

Результаты задач с минимальным уровнем безопасности из выходной очереди доставляются клиентам, а все остальные разделяются на части по алгоритму Шамира и доставляются клиентам по нескольким каналам связи.

**Распределение задач по ресурсам в Cloud системе**

Для оптимальной загрузки ресурсов Cloud системы и для предоставления задачам требуемый уровень сервиса необходимо планировать распределения задач по ресурсам системы. Также необходимо учитывать требования реального времени. Рассмотрим существующие алгоритмы планирования.

**BackFill[8]**

Алгоритм планирования BACKFILL разработан для максимально эффективного использования ресурсов и достижения высокой эффективности системы, предотвращения потенциально чрезмерных задержек, что появляются если начать работать и иметь большую потребность в ресурсах.

Этот алгоритм работает с большим количеством задач, потому ему необходимо большое количество рессусов. В то же время BACKFILL алгоритм может работать в параллельном режиме.

Алгоритм обратного заполнения Backfill работает по следующему принципу: размещая наиболее приоритетное задание, он определяет момент времени, когда освободится достаточное количество ресурсов, занятых уже выполняющимися заданиями, и производит резервирование этих ресурсов. Задание с меньшим приоритетом может быть запущено вне очереди, но только в том случае, если оно не будет мешать запуску всех (в консервативном варианте Backfill) более приоритетных заданий.

Преимущества алгоритма BackFill:

• в условиях работы в приоритетной системе позволяет избежать зависания задания, гарантируя его запуск;

• эффективно загружает ресурсы, позволяя избежать их фрагментации;

• имеет приемлемые временные характеристики при работе на большом количестве вычислительных узлов;

• позволяет работать на множестве гетерогенных ресурсов.

**Min-Min[9]**

Отличительной чертой алгоритма Min-Min является скорость выделения ресурсов для задач. Алгоритм как можно быстрее выделяет ресурсы на каждую из задач, которые могут быть выполнены в кратчайшие сроки. Для этого рассматриваются все задачи, ресурсы которых пока не задействованы. Алгоритм будет выполняться, пока весь набор задач не будет пуст. Задачи заносятся в короткие и длинные очереди, которые параллельно обрабатываются. Главным его недостатком является короткий этап планирования задач на начальных этапах, пока не начнут выполняться длинные задачи.

**Security-Aware Scheduler for Virtual Machines[10]**

Алгоритм используются для размещения виртуальных машин на разных компьютерах, что приводит к риску атак между пользователями внутри виртуальной машины в облачной среде. Такие атаки могут использоваться для нарушения целостности системы. Также, они нарушают конфиденциальность и создают проблемы с VMS. По этому необходимо обеспечивать изоляцию между пользователями в облаке.

Планировщик предоставляет возможность размещения и миграция виртуальных машин на основе политик безопасности, которые позволяют пользователям облака выражать свои потребности приватности в присвоении уровня доступа.

**Алгоритм планирования в Cloud системе A\***

A\* - [алгоритм](http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) поиска [по первому наилучшему совпадению](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%BF%D0%BE_%D0%BF%D0%B5%D1%80%D0%B2%D0%BE%D0%BC%D1%83_%D0%BD%D0%B0%D0%B8%D0%BB%D1%83%D1%87%D1%88%D0%B5%D0%BC%D1%83_%D1%81%D0%BE%D0%B2%D0%BF%D0%B0%D0%B4%D0%B5%D0%BD%D0%B8%D1%8E) на [графе](http://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Для тестирования модели Private Cloud системы используется этот алгоритм, поскольку он обеспечивает высокое качество планирования и позволяет гибко учитывать требования задач к реальному времени.

Порядок обхода вершин определяется [эвристической функцией](http://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D1%80%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0) «расстояние + стоимость» (обычно обозначаемой как *f(x)*). Эта функция — сумма двух других: функций и стоимости достижения рассматриваемой вершины (*x*) изначальной (обычно обозначается как *g(x)* и может быть как эвристической, так и нет) и эвристической оценкой расстояния от рассматриваемой вершины к конечной (обозначается как *h(x)*).

Проиллюстрируем работу алгоритма A\* для задачи о назначениях. Учитывая набор из четырех задач {t0, t1, t2, t3,} и набор из трех процессоров {p0, p1, p2}, найдем результирующие поисковые деревья.

(a)

(b)

Рис. 3 (a) Граф задач, (b) Топология процессоров

Таблица 1

*Матрица стоимостей*

|  |  |  |  |
| --- | --- | --- | --- |
|  | P0 | P1 | P2 |
| T0 | c00 | c01 | c02 |
| T1 | c10 | c11 | c12 |
| T2 | c20 | c21 | c22 |
| T3 | c30 | c31 | c32 |

Узел в поисковом дереве включает в себя частичное распределение задач по процессорам, и значение f (стоимость частичного распределения). Постановка m задач на n процессоров отображается тройкой из m цифр, a0, a1, ..., аm - 1, где ai (0 ≤ i ≤ m - 1) - процессор (от 0 до n - 1) для которого алгоритм присвоил i-ю задачу, значение аi= x указывает, что i-я задача пока не назначена. Каждый уровень дерева соответствует задаче; таким образом, назначение этой задачи на процессор заменяет значение x в строке назначения некоторым номером процессора. Корневой узел включает в себя набор всех свободных задач xxxx. Далее мы рассматриваем стоимость размещения задачи t0 на p0 (0xxx), t0 на p1 (1xxx) и t0 на p2 (2xxx). Назначение t0 на p0 (0xxx) приводит к общей стоимости *f*(n) = *g*(n), в данном случае, равно c00, что является стоимостью выполнения t0 на p0. Аналогично рассчитываем затраты на назначение t0 на p1 (c01) и t0 на p2 (c02). Алгоритм добавляет эти три узла в список обработанных, а также выбирает узел с минимальной стоимостью для расширения.

Поиск продолжается до тех пор, пока процесс не выберет узел с полным назначением для расширения.

**Модифицированный алгоритм A\***

Модифицированный алгоритм A\*использует значительно меньше памяти, чем O \* и \* R и работает параллельно.

Параллельный алгоритм направлен на ускорение поиска с помощью параллельной обработки. Это делается путем деления дерево поиска среди процессорных элементов (ПЭ) как можно более равномерно и отбрасыванием несущественных узлов.

Первоначально, мы статически разделим дерево поиска в зависимости от количества ПЭ P в системе и максимального количества S наследников узла в дереве поиска. Существуют три способа начального разбиения:

* При P < S. Каждый ПЭ расширяется только начальным узлом, что порождает S новых узлы. Каждый ПЭ получает один узел, дополнительные узлы распределяются циклически (RR).

• P = S. Каждый ПЭ расширяется только начальным узлом, и каждый ПЭ получает один узел.

• P> S. Каждый ПЭ продолжает расширять узлы, начиная от начального узла (нулевое задание), пока порядковый номер узлов меньше P. Мы отсортируем список в порядке возрастания стоимости узлов. Первый узел в списке идет наПЭ1, второй узел к ПЭp, третий узел ПЭ2, четвертый узел ПЭp-1, и так далее. Дополнительные узлы распределяются циклически (RR). Хотя это распределение не гарантирует, что лучшая стоимость узла на начальных уровнях дерева приведет к лучшей стоимости узел в дальнейшем, алгоритм продолжает равномерно распределять хорошие узлы настолько равномерно, насколько возможны между всеми ПЭ.

Если поиск находит решение, алгоритм завершается.

После первоначального статического разделения, каждый ПЭ будет искать только в своем поддереве. Некоторые ПЭ могут работать схорошей части поискового пространства, в то время как другие могут расширять ненужные узлы. Чтобы избежать этого, ПЭ необходимо общаться друг с другом, поделить лучшую часть поискового пространства и избежать ненужной работы. Эта связь может быть глобальной ( ПЭ транслирует свои узлы на все другие элементы) или местной (ПЭ общается только со своими соседями).

При первоначальном разделении ,каждый ПЭ имеет один или несколько узлов в списке OPEN. ПЭ определяет своих соседей с помощью процессорной сетки и своих х и у координат, периодически выбирает соседа и посылает ему свой лучший узел. Помимо балансировки нагрузки, ПЭ также передает свое решение (когда он его находит) всем ПЭ. Это помогает избежать лишней работы для ПЭ, которые работают над плохой частью поискового пространства. После того, как ПЭ получает лучшее решение по сравнению со своим лучшим узлом, оно станавливает расширение ненужных узлов ПЭ, который находит первое решение транслирует свою стоимость на все другие элементы. После этого, ПЭ передает стоимость только в том случае, если она лучше полученных ранее.

**Модификация алгоритма для применения в Real Time системе**

Для работы в условиях реального времени алгоритм модифицирован. Каждый ресурс имеет коэффициент загрузки kload в диапазоне [0,1] – процент занятого процессорного времени. При загрузке задачи на ресурс, рассчитывается какой процент процессорного времени задача требуетk’load. Данный коэффициент вводится для того, чтобы избежать возможной перегрузки узла. Поскольку при перегрузке не гарантируется заданное время отклика системы.

Если kload+k’load>1, значит ресурс будет перегружен и не сможет соответствовать требованиям реального времени, потому загрузка этой задачи на этот ресурс невозможна. Значение функции стоимости для данного узла устанавливается в бесконечность.

Иначе коэффициент загрузки ресурса, после погружения задачи на этот ресурс, будет равен:

kload=kload+k’load

Также учитывается уровень безопасности задачи и ресурса. Если уровень безопасности ресурса выше или равен уровню безопасности задачи, значит выполнения задачи на этом ресурсе возможно, иначе ресурс для задачи не подходит и значение функции стоимости устанавливается в бесконечность. Это позволяет обеспечить требуемый уровень безопасности для задачи во время выполнения.

Таким образом модификация алгоритма планирования позволяет избежать перегрузки системы и обеспечивает требуемый уровень безопасности выполнения задач.

**Разделение данных для безопасной передачи по открытому каналу**

Для безопасной передачи данных по открытым каналам связи можно использовать разделение данных, для передачи частей по разным каналам. Для восстановления исходного сообщения, разделенного на n частей, необходимо собрать k частей. Таким образом безопасность передачи повышается.

**Алгоритм Блекли [7]**

С помощью алгоритма Блекли можно создать (n,k)-алгоритм разделения секрета для любых n и k: для этого надо положить размерность пространства равную n, и каждому из n игроков дать одну гиперплоскость, проходящую через секретную точку. Тогда любые k из n гиперплоскостей будут однозначно пересекаться в секретной точке.

В этой схеме используются простые числа. Для (n,k) пороговой схемы выбирается простое число p, большее M. Затем выбираются числа меньше p: d1,d2,…,dn, для которых:

1. Значения di упорядочены по возростанию

2. Каждое di взаимно-простое с другими di

3. d1\*d2\*…\*dk>p\*dn-k+2\*dn-k+3\*…\*dn

Чтобы распределить части, сначала выбирается случайное число r и вычисляется M’=M+rp

Тенями ki являются ki=M’ mod di

Объединив любые k частей можно восстановить M, используя теорему об остатках, но это невозможно сделать используя k-1 частей.

**Алгоритм Karnin-Greene-Hellman [7]**

В этом алгоритме используется матричное умножение. Выбирается n+1 k-мерных векторов V0,V1,…,Vk так, что ранг любой матрицы размером k\*k, образованный из этих векторов равен k. Вектор U это вектор размерности k+1.

M-это матричное произведение U\*V0. Частями являются произведения U\*Vi, где i меняется от 1 до k.

Любые k частей можно использовать для решения системы линейных уравнений размерности k\*k, неизвестными являются коэффициенты U. U\*V0 можно вычислить по U. Используя любые k-1 частей, решить систему уравнений и таким образом получить секрет невозможно.

**Алгоритм Шамира [6]**

Идея Алгоритм заключается в том, что двух точек достаточно для задания прямой, трех точек — для задания параболы, четырёх точек — для кубической параболы, и так далее. Чтобы задать многочлен степени *k* требуется *k+1* точек.

Для разделения данных таким образом, чтобы восстановление было возможно при наличии k частей, данные трансформируются в формулу многочлена степени. Восстановить этот многочлен можно по точкам. Количество же различных точек многочлена не ограничено.

Алгоритм Блекли менее эффективна, чем алгоритм Шамира: в алгоритме Шамира каждая доля такого же размера как и секрет, а в алгоритме Блекли каждая доля в k раз больше. Поэтому ми в нашей роботе используем алгоритм Шамира. Рассмотрим алгоритм Шамира детальней.

Выберем некоторое [простое число](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) p>M. Это число открыто сообщается всем участникам. Оно задаёт [конечное поле](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) размера р. Над этим полем строится многочлен степени k-1 (то есть случайно выбираются все коэффициенты многочлена, кроме М):



В этом многочлене М — это разделяемый секрет, а остальные коэффициенты

 — некоторые случайные числа, которые уничтожаются после того, как процедура разделения секрета будет завершена.

Далее вычисляются координаты различных точек:



Номера секретов должны быть различны по модулю p.

После этого секреты вместе с их номером, числом и степенью многочлена передаются сторонам. Случайные коэффициенты удаляются. Теперь любые участники, зная координаты k различных точек многочлена, смогут восстановить многочлен и все его коэффициенты, включая последний из них — разделённый секрет.

Прямолинейное восстановление коэффициентов многочлена через решение системы уравнений заменяется на вычисление [интерполяционного многочлена Лагранжа](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD_%D0%9B%D0%B0%D0%B3%D1%80%D0%B0%D0%BD%D0%B6%D0%B0). Формула многочлена будет выглядеть следующим образом:





Где xi, xj— координаты точек многочлена. Все операции выполняются также в конечном поле.

**Применение алгоритма Шамира в Cloud системах реального времени**

Для обеспечения безопасности доставки задач до cloud системы и доставки результатов клиенту целесообразно использовать алгоритм разделения секрета Шамира.

Для этого на клиенте устанавливаются программные средства для разделения задачи и задача отправляется по n каналам связи во входную очередь Cloud системы, где и собирается из частей.

**Масштабируемость**

Масштабируемость имеет решающее значение для успеха многих организаций, которые в настоящее время ведут бизнес в Интернете.

При возрастании нагрузки на сервисы данного предприятия, появляется необходимость выделения больших ресурсов, которые смогут обслуживать заявки пользователей. Как правило, под понятием “ресурсы” можно подразумевать серверы, процессоры, устройства хранения данных, пропускную способность сети.

**Показатели масштабируемости**

Несмотря на то, что различные сервисы и веб-приложения могут работать по-разному, можно выделить несколько общих точек масштабирования, в которых ресурсы становятся ограниченными. Например, очень маловероятно, что веб-приложение может расти и справляться с возрастающей нагрузкой на одном сервере. В какой-то момент времени, необходимо добавить дополнительные серверы для удовлетворения растущих потребностей в целях сохранения качества обслуживания. Тоже правило справедливо и для устройств хранения данных, сети и других точек масштабирования.

Важно понять, как веб-приложение потребляет ресурсы, как оно ведет себя при высокой нагрузке, и что происходит в потенциальных точках масштабирования всей системы в целях поддержания желаемой производительности. Для этого можно выделить некоторые типичные показатели масштабирования:

* Количество одновременно работающих пользователей
* Количество активных соединений
* Количество запросов в секунду
* Среднее время отклика на запрос

После обработки показателей и на основе статистических данных, собранных за определенный период времени, принимается решение об увеличении или уменьшении количества активных сущностей данного приложения.

**Масштабируемость Real Time систем**

При работе с Real-Time системами, должен учитываться тот факт, что любой момент времени данная система должна быть доступна без простоев и должна обеспечить мгновенное время реагирования, независимо от количества одновременно работающих пользователей.

Основная проблема с такими системами заключается в отсутствии способности планирования количества пользователей, которые будут обращаться к сервису. Решение состоит в том, чтобы динамически масштабировать приложение и позволить количеству серверов и прочих компонентов расти (или уменьшаться) по требованию. Масштабируемая облачная архитектура, которая эффективно решает данный сценарий, показанана рис.4. Данная архитектура включает в себя интерфейсный балансировщик нагрузки, некоторое количество приложений на виртуальных машинах, подсистему резервирования и модуль мониторинга нагрузки с динамическим алгоритмом масштабирования.

Балансировщик нагрузки

Виртуальная машина 1

Виртуальная машина N

\*\*\*

Алгоритм масштабирования

Модуль мониторинга

Модуль прогнозирования

**Cloud система**

Рисунок 4 Масштабируемая облачная архитектура

**Балансировщик нагрузки**

Использование подсистемы балансировки нагрузки позволяет маршрутизировать входящие запросы на свободные участки системы. Конфигурация данного модуля может обновляться в реальном времени, что позволяет системе автоматически и динамически добавлять экземпляры приложения и запускать их на новых серверах.

**Модуль мониторинга и резервирования**

Подсистема мониторинга выполняет функцию накопления отдельных показателей масштабирования со всех экземпляров приложения и компонентов системы. На основе этих данных алгоритм динамического масштабирования принимает решение о вызове события в подсистеме резервирования. В зависимости от обновленных статистических данных, происходит масштабирование системы вверх или вниз, а также вносятся изменения в конфигурационный файл балансировщика нагрузки.

**Развертывание на основе эталонных образов**

Запуск новой копии данного приложения на сервере происходит путем развертывания и активации эталонного виртуального образа системы. Одной из проблем при клонировании виртуальных образов является обработка специфических настроек системы. Автоматизация подготовки новых виртуальных машин достигается путем добавления внешних и внутренних модулей в эталонный образ, которые управляют развертыванием.

**Алгоритм масштабирования**

Алгоритм динамического масштабирования, основан на показателе масштабирования Аi в каждом экземпляре виртуальной машины.

На первом шаге алгоритм определяет приложения с числом активных сеансов выше или ниже заданных пороговых значений. Если количество активных сессии на всех экземплярах выше данного верхнего порога, то новый экземпляр приложения будет развернут, запущен и добавлен в интерфейс балансировщика нагрузки. Если имеются случаи с числом активных сеансов ниже заданного нижнего порога и, по крайней мере один из экземпляров не имеет активных сеансов, то он будет удален из балансировщика нагрузки и отключен от системы.

В каждом случае, коэффициенты нагрузки для всех активных экземпляров будут пересчитаны и применены к балансировщику для перераспределения равномерной нагрузки.

**Модификация алгоритма для эффективной работы в условиях реального времени и уровней доступа**

В условиях реального времени на первый план выходит время отклика и выполнения задачи. Потому необходимо учитывать что время переноса задачи на другую машину, а также время запуска нового экземпляра, могут отразится на времени отклика. Для этого предлагается ввести для задачи параметр – время отклика. И при возникновении ситуация, когда нужно переместить задачу на другую машину, проводится расчет времени перезапуска задачи на машине, если это время больше установленного для задачи времени отклика, то такой перезапуск проводить нельзя. И эта операция не производится.

Поскольку более приоритетные задачи требуют более высокого уровня сервиса, то и требования к ресурсам различны. Потому нужно дифереинцировать уровни пороговых значений для каждого приоритета. Для этого предлагается ввести два массива значений: верхнего и нижнего предела для каждого уровня приоритета, таким образом можно гибко учесть требования задач к уровню обслуживания.

|  |
| --- |
| For i:=0 to N{  Определить загрузку i-ой машины – MachineLoad;  If(machineLoad>Mupper[уровень доступа машины]){  Запустить новый экземпляр машины с таким же уровнем доступа;  }  If(machineLoad<Mlower[уровень доступа]){  RemovedTaskCount:=0;  For j:=0 to Mi{  Расчитать время перезапуска задачи T[j] – TaskRebootTime;  If(TaskRebootTime<TaskBorderTime[j]){  Перезапустить задачу на другой машине;  Удалить задачу из машины;  RemovedTaskCount++;  }  If(RemovedTaskCount== Mi){  Остановить машину i;  }  }  Где:  TaskBorderTime[]:массив времен отклика для всех задач Mupper[]:Верхний порог нагрузки Mlower[]:Нижний порог нагрузки N-количество машин  Mi-количество задач на i-ой машине |

Рисунок 5 Алгоритм балансировки нагрузки в Cloud системах реального времени

**Моделирование системы**

Одной из важнейших характеристик моделирования, произвольной вычислительной системы, являются вычислительные ресурсы необходимые для его проведения. К наиболее существенным ресурсам следует отнести объем требуемой оперативной памяти ВС на которой производится моделирование и время, необходимое для его проведения. Если вышеназванные параметры, превышают возможности инструментальной вычислительной системы, то решение задачи моделирования невозможно, либо достигается оно за неприемлемое для пользователя время.

Основной целью проводимого вычислительного эксперимента, являются: исследование влияния многоканальности на работу Cloud. Ограниченный набор воздействий и исследуемых характеристик создает предпосылки для упрощения используемой модели, а тем самым и к уменьшению ресурсов необходимых для моделирования. При этом точность определяемых параметров ухудшается только незначительно.

Создание модели, учитывающей все свойства вычислительной системы возможно, однако ее решение требует чрезвычайно больших вычислительных ресурсов. Поэтому, с целью проведения процесса моделирования, следует применить упрощенную (приближенную) модель, сложность решения которой приемлема, а точность достаточно высока.

Для получения высокого качества моделирования, разработано программное обеспечение для моделирования в Cloud системе. Это позволяет гибко настраивать вычислительные мощности.

Для моделирования системы использовалось 100 потоков Эрланга, каждый из которых моделирует поведения пользователя. Согласно распределения потока Эрланга генерировалось поступление задач во входную очередь и параметры этих задач.

Модель Private Cloud системы состоит из 20 машин.

Снимались такие характеристики системы:

- Зависимость загрузки узла от количества задач во входной очереди;

- Зависимость времени ожидания задачи от количества задач во входной очереди

- Зависимость времени ожидания задачи от группы безопасности-приоритета пользователя;

- Зависимость длины входной очереди от времени;

- Зависимость количества отброшенных задач от длины входной очереди;

- Зависимость вероятности отказа выполнения задачи от длины входной очереди.

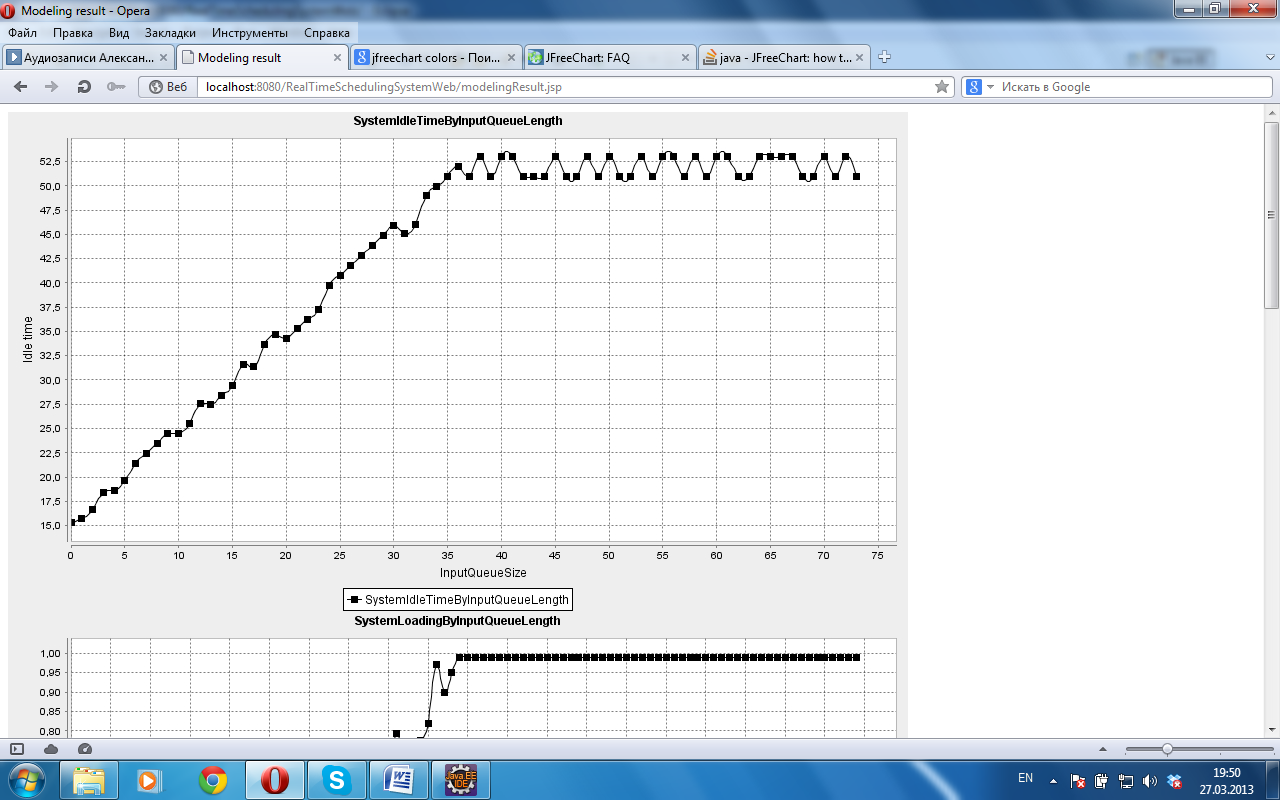
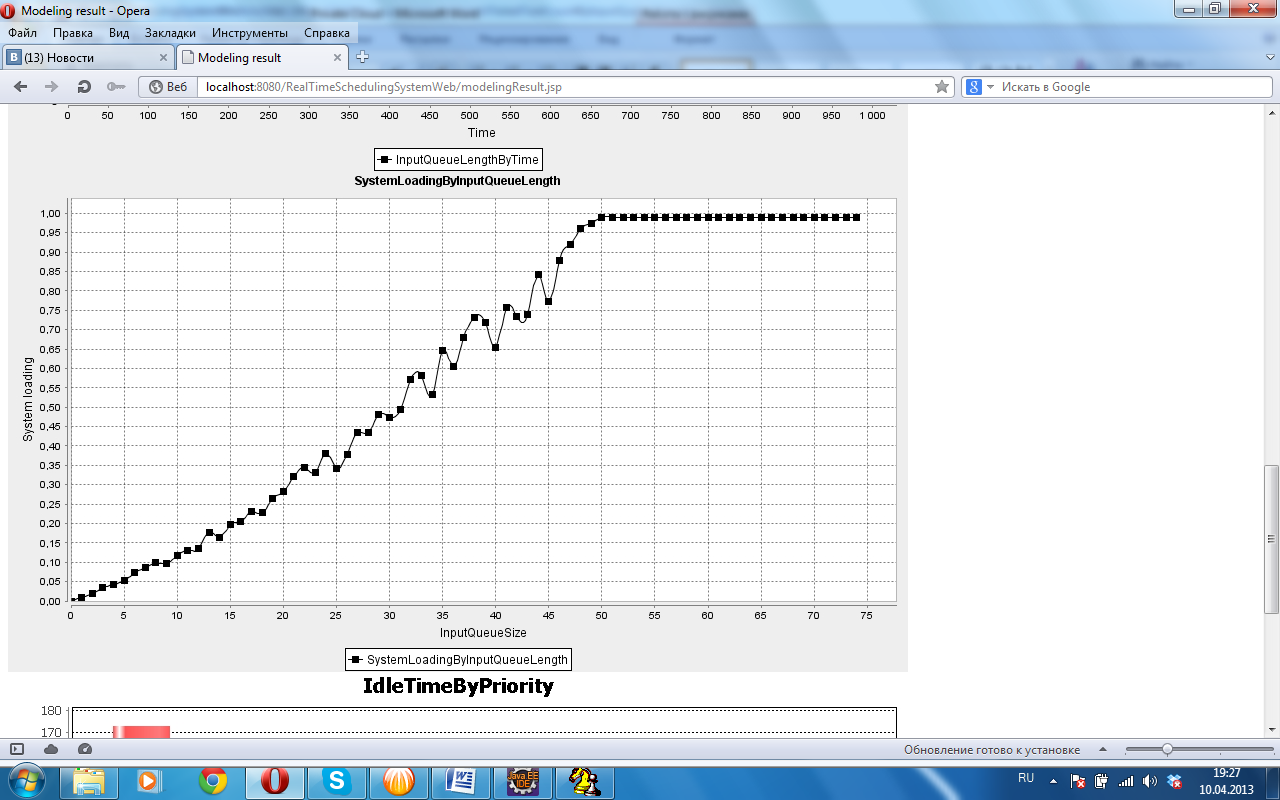


Рис.6 График зависимости ожидания завершения выполнения задачи от длинны входной очереди

Как видно из рисунка 6, время ожидании задачи растет вместе с ростом длины входной очереди, это объясняется тем, что увеличивается время простоя в входной очереди и большей загрузкой узлов. При достижении полной загрузки, система отбрасывает новые входящие задачи, поскольку при приеме большего количества задач, система не сможет удовлетворять требованиям систем реального времени.

Минимальное время ожидания, равно времени выполнения задачи.

  
Рис. 7 График зависимости загрузки системы от длинны входной очереди

На рисунке 7 видно, что при увеличении количества задач во входной очереди, загрузка системы растет и когда наступает 100 процентная загрузка, задачи в систему не поступают. За счет отброса задач при достижении системы пиковой нагрузки, обеспечиваются требования реального времени.

Нелинейность роста графика объясняется тем, что из-за продолжительного времени обработки задачи, скорость обработки меньше скорости поступления задач во входную очередь. Тем самым, загрузка системы растет нелинейно.

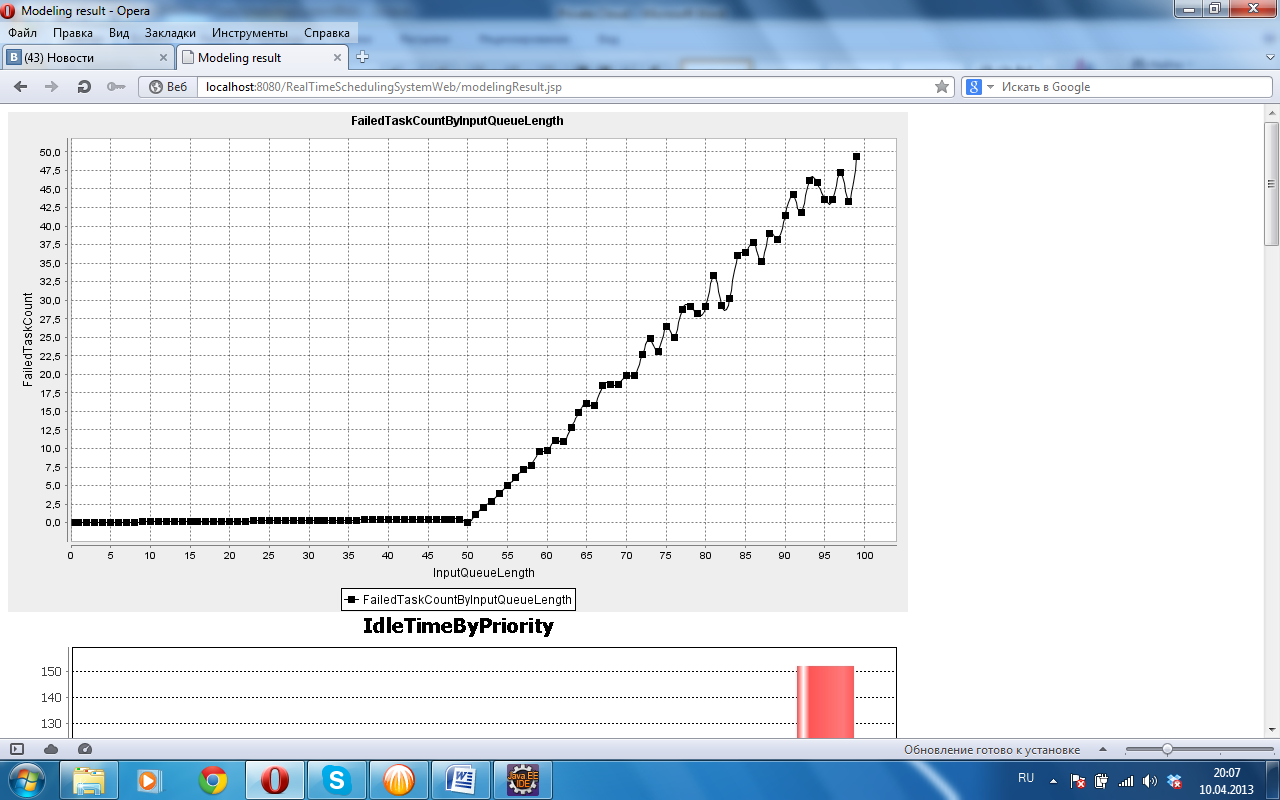


Рис. 10 График зависимости количества отброшенных задач от длины входной очереди

Как видно из рисунка 10, при количестве задач в входной очереди до 50, система обрабатывает все задания. После 50, количество отбрасываемых задач возрастает линейно и равно N-50, где N – количество задач во входной очереди. Это объясняется тем, что при N~=50 наступает 100% загрузка системы, то есть система одновременно может обрабатывать 50 задач. Количество задач для полной загрузки системы можно определить из рисунка 5.

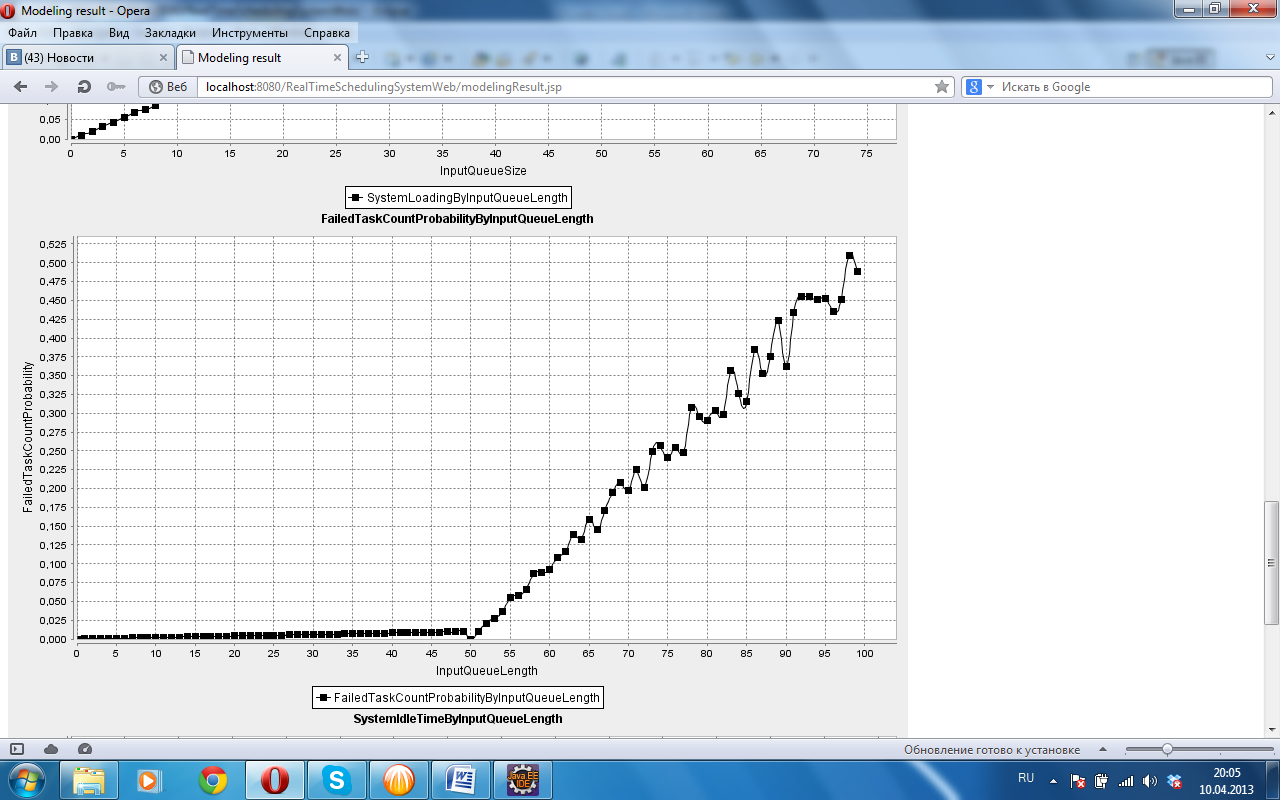


Рис. 11 График зависимости вероятности отброса задачи от размера входной очереди

Вероятность отказа p рассчитывается по формуле , где K- количество отброшенных задач, N-количество выполненных задач.

Как было описано выше полная загрузка системы наступает при 50 задачах, то есть система может одновременно выполнять около 50 задач. То есть при количестве задач во входной очереди меньше 50, вероятность отброса задачи равна 0, а потом она растет линейно, с ростом количества задач во входной очереди.

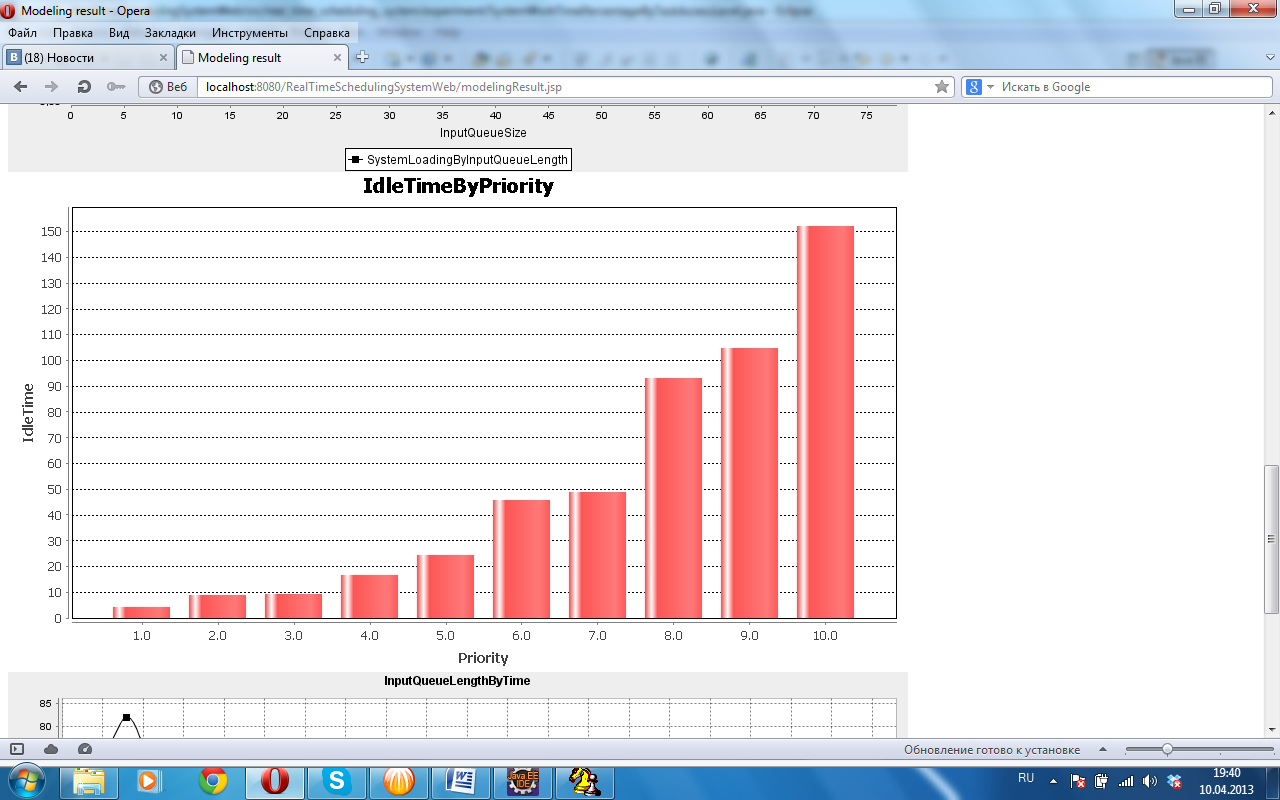


Рис. 8 График зависимости времени ожидания задачи от уровня приоритета задачи

Из графика 8 видно, что задачи с более высоким приоритетом проводят времени в ожидании меньше чем задачи с низким приоритетом. Чем выше загрузка системы, тем выше время ожидания для низкоприоритетных задач.

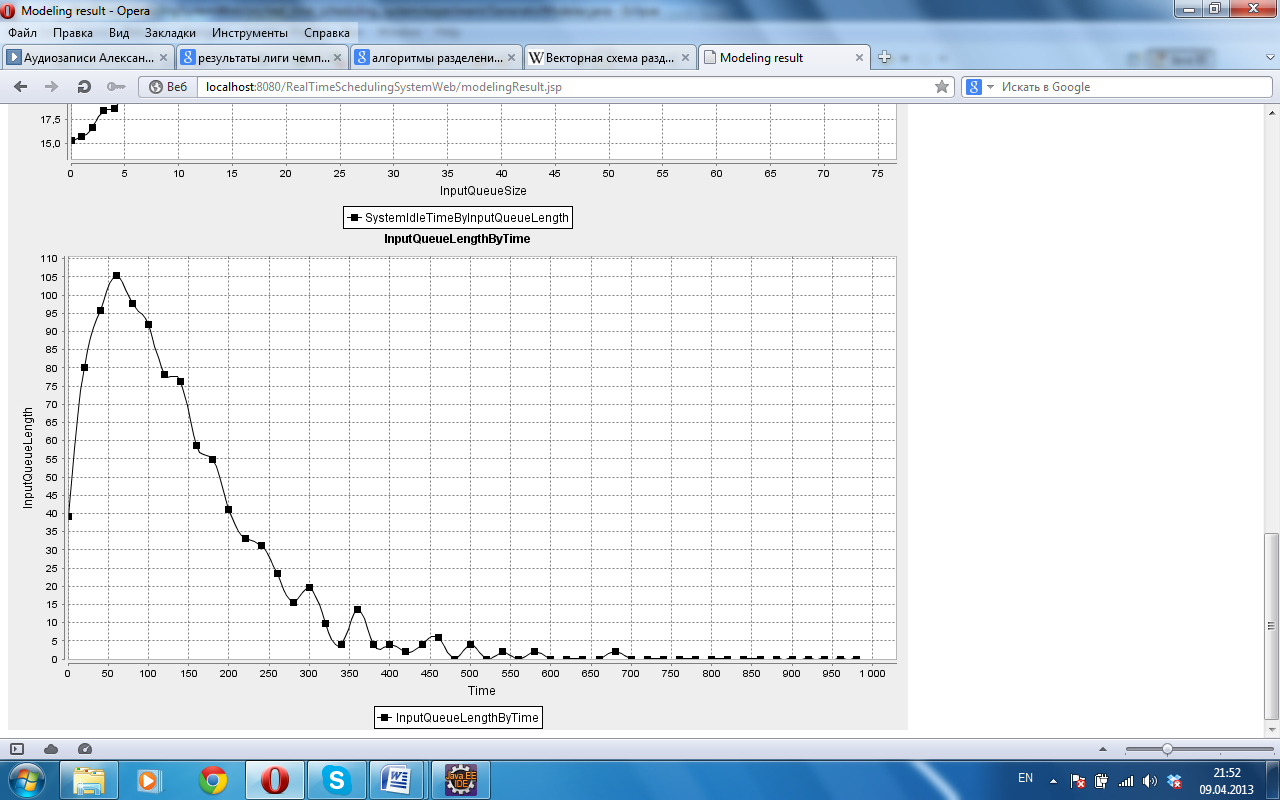


Рис. 9 График зависимости длины входной очереди от времени

Моделирование входного потока заявок проводилось сложением 100 потоков Эрланга, на выходе получается поток Эрланга, поскольку в обращении клиентов наблюдается периодичность, моделирование входной очереди также проводится с периодом.

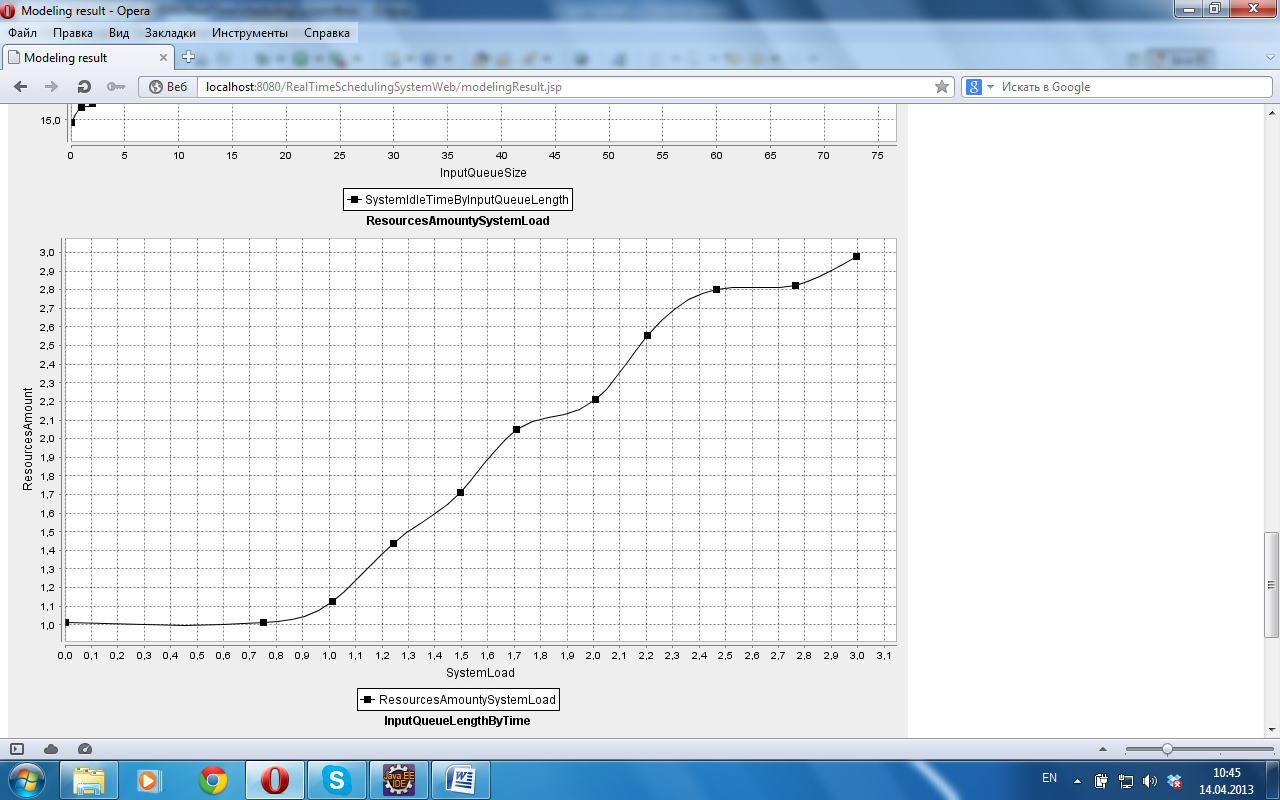


Рисунок 12 График зависимости количества ресурсов от уровня загрузки системы

Вертикальная ось – количество ресурсов в системе, относительно начального количества.

Горизонтальная ось – загрузка системы относительно начальной загрузки.

При достижении системой уровня загрузки больше порога нагрузки начинают подключатся дополнительные ресурсы. На графике видно ступенчатое изменение количества ресурсов, это обьясняется тем, что ресурсы подключаются дискретными частями.

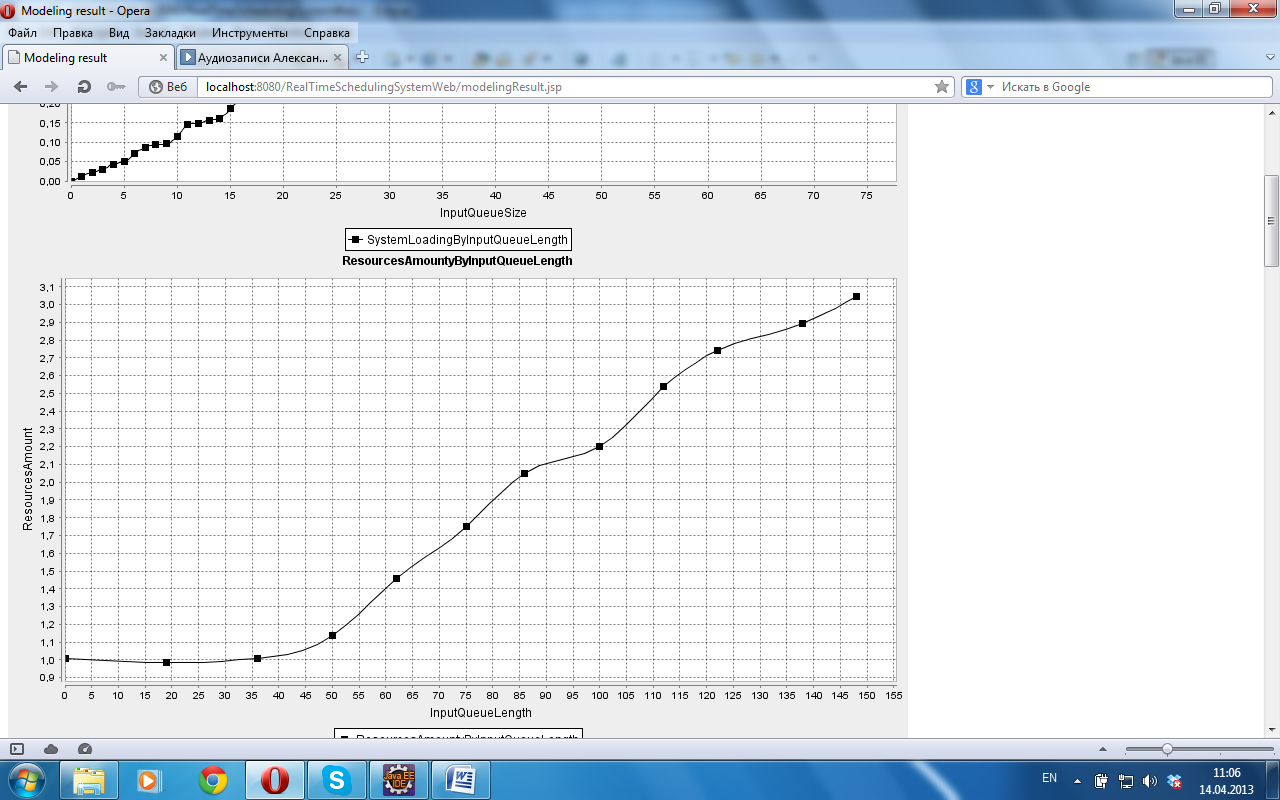


Рисунок 13 График зависимости количества ресурсов от длины входной очереди

Вертикальная ось – количество ресурсов в системе, относительно начального количества.

Горизонтальная ось – количество задач во входной очереди.

Длина входной очереди определяет загрузку системы. Потому характер графика совпадает с графиком зависимости количества ресурсов от уровня загрузки системы.

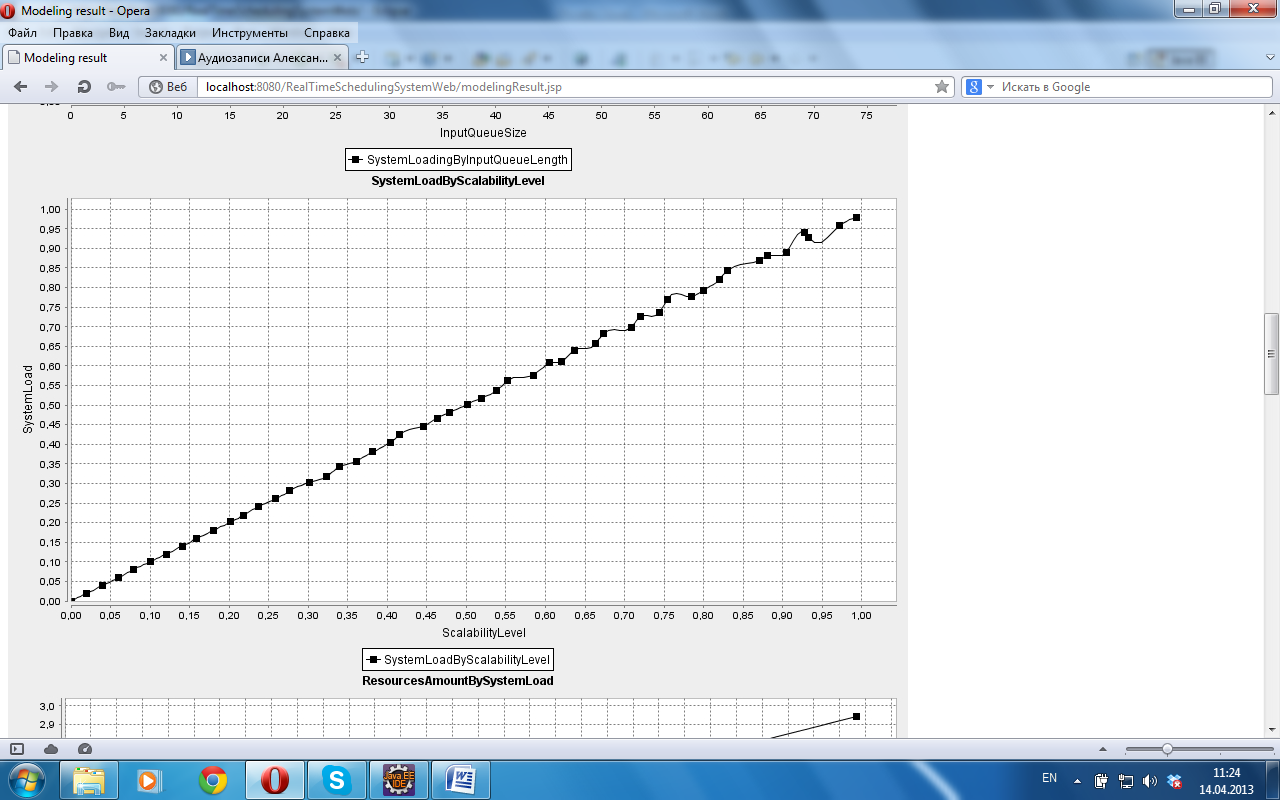


Рисунок 14 График зависимости загрузки от среднего порога нагрузки

Средний порог нагрузки – среднее арифметическое порогов нагрузки для всех уровней приоритета. Порог нагрузки определяет уровень загрузки для уровня приоритета после достижения которого, происходит подключение дополнительных ресурсов.

Как видно из графика, загрузка системы растет при увеличении порога, поскольку чем меньше порог, тем раньше начинают выделяться ресурсы.

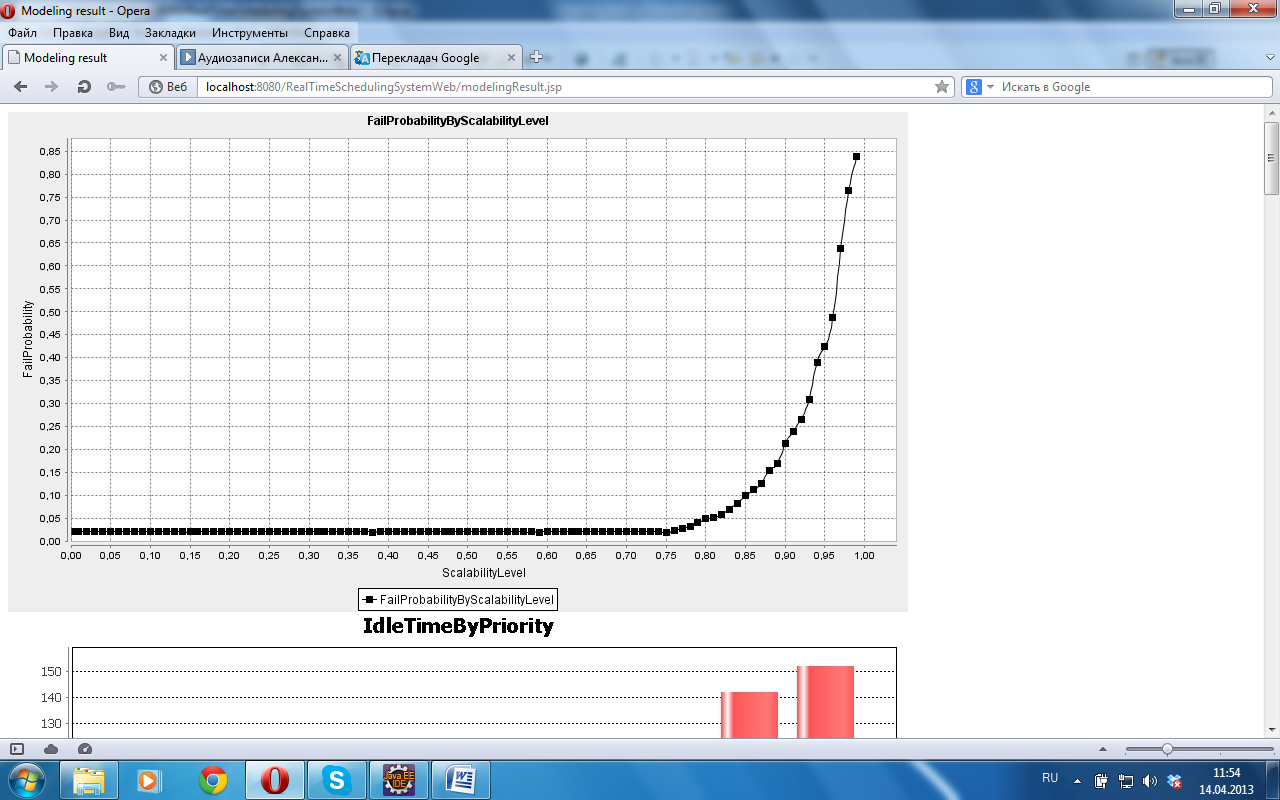


Рисунок 15 График зависимости вероятности отказа от среднего порога назгрузки

Вероятность отказа p рассчитывается по формуле , где K- количество отброшенных задач, N-количество выполненных задач.

При низком пороге нагрузки, ресурсы начинают выделятся при низкой загрузке системы потому вероятность отброса задачи низкая, при достижении точки, когда очередная задача может перегрузить машину вероятность отброса задачи быстро растет, поскольку подключить дополнительные ресурсы за время, которое требует новое задача – невозможно, приходится отбрасывать задачу.

Созданная модель Private Cloud системы реального времени доступна по адресу <http://rtss.elasticbeanstalk.com>. Выполнение моделирования в Cloud системе позволяет справляться с высокими требованиями к вычислительным ресурсам и динамической нагрузкой.

**Выводы.** Был рассмотрен алгоритм планирования выполнения задач в cloud системе A\*, адаптирован и модифицирован для работы в Cloud системе реального времени. Модуль, использующий этот алгоритм, выполняет функции распределения задач и балансировки нагрузки. Алгоритм позволяет задавать уровень безопасности и задача будет выполнятся только на машинах, которые имеют уровень безопасности не ниже указанного. Это позволяет обезопасить данные в момент выполнения задачи.

Для обеспечения необходимого уровня безопасности при передаче от клиента в систему и назад, использован алгоритм разделения секрета Shamir. Если пользователь требует уровень безопасности выше минимального, то задача и данные разделяются на части и передаются в Cloud систему по отдельным каналам, где собираются обратно. Это позволяет повысить уровень безопасности при передаче.

Результаты моделирования показали, что Cloud система в нормальном режиме (с уровнем загрузки меньше 100%) удовлетворяет требованиям реального времени. Как видно из рисунка 9, вероятность отброса задачи приблизительно равна 0, до достижением системы пика загрузки.

Также разработанная система приоритетов позволяет эффективно дифереенцировать пользователей по уровню обслуживания. Как видно из рисунка 6, задачи с уровнем 1 (наиболее приоритетный уровень) имеет наиболее низкое время ожидания.

**Список литературы:**

1.National Institute of Standards and Technology<http://www.nist.gov/index.html>

2.[Smart Computing in Real Time](http://real-timecloud.com/2013/01/17/smart-computing-in-real-time/) <http://www.real-timecloud.com>

3.CloudComputingJournal <http://cloudcomputing.sys-con.com/>

4.Heinrich-Hertz-Institut <http://www.hhi.fraunhofer.de>

5.Optimal Task Assignment in Heterogeneous Distributed Computing Systems - Muhammad Kafil and Ishfaq Ahmad, The Hong Kong University of Science and Technology

6. Shamir, Adi (1979), "How to share a secret", Communications of the ACM

7. Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — С. 589. — 816 с. — 3000 экз. — ISBN 5-89392-055-4

8. Использование алгоритма Backfill в грид Коваленко В.Н., Семячкин Д.А. Труды международной конференции "Распределенные вычисления и Грид-технологии в науке и образовании" (Дубна, 29 июня-2 июля 2004 г.).- Дубна: 11-2004-205, ОИЯИ, 2004, сс. 139-144

9. M. Dorigo, 1992. Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italy.

10. A Security-Aware Scheduler for Virtua lMachines on IaaS Clouds, ZainaAfoulki, AlineBousquet, Jonathan Rouzaud-Cornabas, Rapport de Recherche, 2011