1. **Private Cloud**

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.

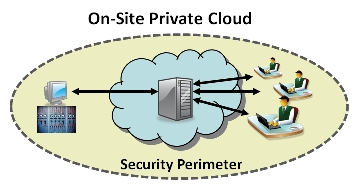


Рисунок 1 Собственное частное облако

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.

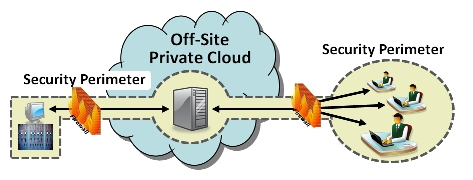


Рисунок 2 Стороннее частное облако

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.

**2. Основные требования для облачных систем реального времени [2,3,4]**

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. Доставка данных методом публикаций / подписок. В событийной модели пользователь регистрирует запрос на получение данных один раз, а затем получает обновления по мере их появления.

3. 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.

4. Обратный тип взаимодействия клиент / сервер. Обычно система управления на предприятии выступает в качестве сервера (как источник данных), а облачный сервис в качестве клиента, который обращается к этим данным. Это означает, что непосредственно на предприятии необходимо держать открытым порт брандмауэра. Для обеспечения безопасности облачных систем необходимо обратное взаимодействие клиент / сервер, то есть система на предприятии должна выступать в качестве клиента, а облачный сервис в качестве сервера, даже если система является источником данных. Это позволяет системе на предприятии передавать данные в облако не компрометируя себя.

5. Инфраструктура должна ориентироваться на данные, а не на веб сервисы, дизайн. Данные остаются в своем простом формате, без кода HTML или XML, для обеспечения минимально возможной задержки.

6. Прямой доступ кraw данным в облаке. Данные от источника должны проходить через облако к конечному пользователю в raw формате, а преобразование в другие форматы (такие как HTML, XML, SQL и т.д.) должно происходить в последний момент.

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

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

9. 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.

**3. Алгоритм планирования в Cloud системе A\* [5]**

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).

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 пяти задач {t0, t1, t2, t3, t4} и набор из трех процессоров {p0, p1, p2} (см. Рис 1), найдем результирующие поисковые деревья.

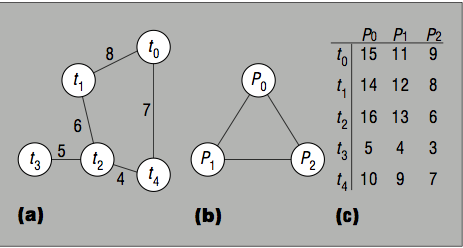


Рисунок 3 (a) Граф задач, (b) Топология процессоров,(c)Матрица стоимостей

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 строкой 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 XXXXX.Далее, для примера на рисунке 2, мы рассматриваем, стоимость размещения задачи t0 на p0 (0XXXX), t0 на p1  (1XXXX) и t0 на p2 (2XXXX). Назначение t0 на p0 (0XXXX) приводит к общей стоимости *f*(n) = 30. *g*(n), в данном случае, равно 15, что является стоимостью выполнения t0на p0. *h*(n) также равно 15, что является суммой минимального выполнения или стоимостью связи t1 и t4 (задачи обмениваются данными с t0). Аналогично рассчитываем затраты на назначение t0на p1 (26) и t0на p2 (24)–минимальная стоимость. Алгоритм добавляет эти три узла в список OPEN, а также выбирает узел 2XXXX для расширения.

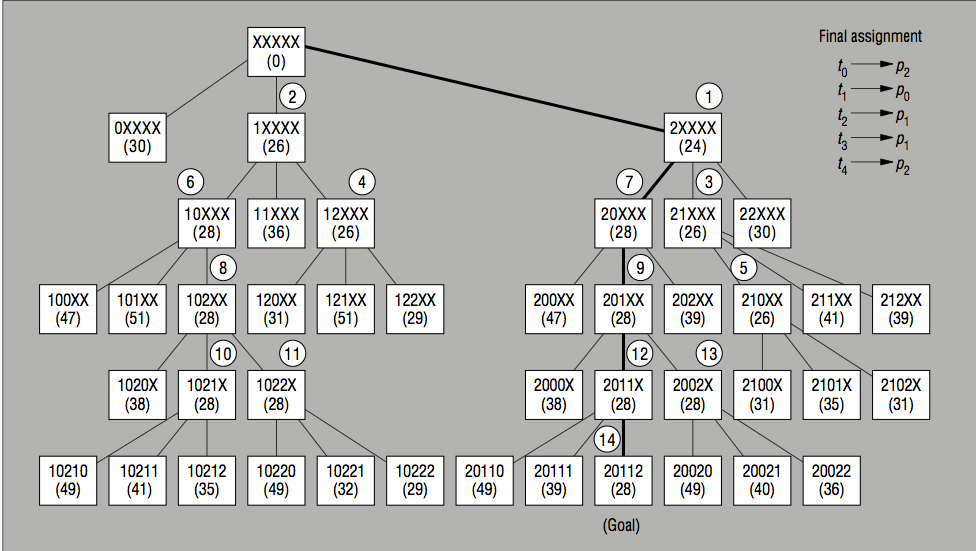


Рисунок 4 Алгоритм A \* O

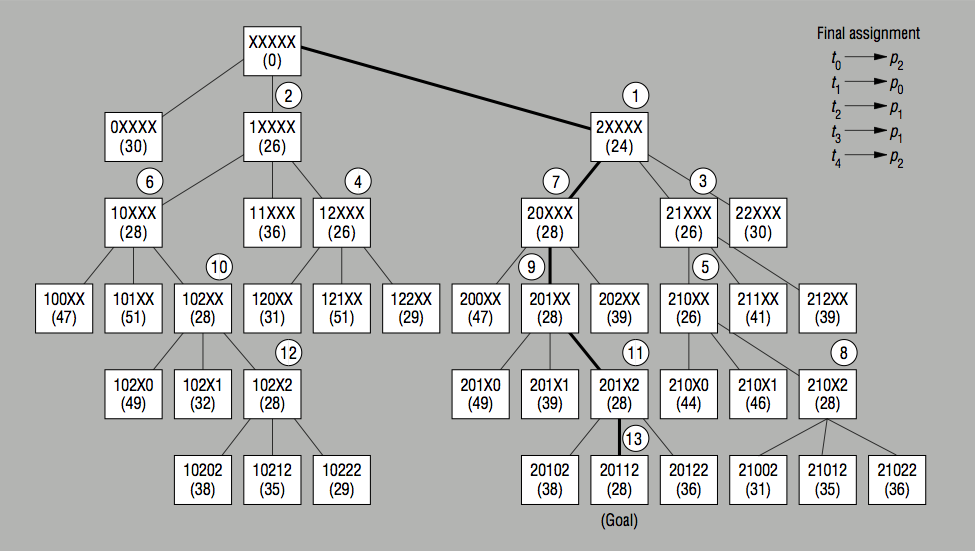


Рис 5.Алгоритм А\* R

Алгоритм расширяет узел 2XXXX следующим образом. По второму уровню дерева, алгоритм будет рассматривать t1дляназначения и 20XXX, 21XXX, и 22XXX три возможных варианта. Значение*f* (n) для 20XXX составляет28, и вычисляется следующим образом: выбирается процессор с наибольшей нагрузкой (p0 в этом случае). *g* (n) = 22, что является стоимость выполнения t1на p0 (14) плюс расходы на связь между t1 и t0 (8), поскольку они назначены на два разных процессора. *h* (n) = 6, стоимость минимального выполнения или связи t2 (единственная неназначенная задача, которая взаимодействует с t1). Аналогично вычисляются значения*f* (n) для 21XXX и 22XXX. На данный момент, узлы 0XXXX, 1XXXX, 20XXX, 21XXX, и 22XXX находятся в списке OPEN. Поскольку узел 1XXXX имеет минимальную стоимость, алгоритм расширяет его рядом, после чего получаем узлы 10xxx, 11xxx, и 12XXX. Поиск продолжается до тех пор, пока процесс не выберет узел с полным назначением (20112) для расширения. В этот момент, этот узел является целевым, т.к. имеет полное назначение и минимальную стоимость.

На рисунке 2, порядок котором алгоритм рассматривает задачи для присваивания такой:{t0, t1, t2, t3, t4}, а вовремя поиска оптимального решения создаются 42 узла и 14 расширяются. Как показано на рисунке 3, модифицированный алгоритм рассматривает последовательность {t0, t1, t2, t4, t3}.В этом случае генерируется 39 узлов и 13 узлов расширяются. Оптимальное назначение 20112, с той же оптимальной стоимостью решения (28).

**3.1 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.

**3.2 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.

**4. Алгоритм разделения секрета Шамира**

Алгоритм разделения секрета Шамира позволяет разделить секретную информацию на n частей, таким образом, чтобы при наличии (t-1) частей, ее восстановление было невозможно, а при наличии t и более частей информацию можно было восстановить.

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 p. Над этим полем строится многочлен степени k-1(то есть случайно выбираются все коэффициенты многочлена, кроме M):

 (1)

В этом многочлене M — это разделяемый секрет, а остальные коэффициенты a_{k-1}, a_{k-2}, \dots, a_1 — некоторые случайные числа, которые уничтожаются после того, как процедура разделения секрета будет завершена.

Далее вычисляются координаты различных точек:

(2)

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 kразличных 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:

(3)

(4)

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

**4.1 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.

**5. Описание модели СРВ**

Группа клиентов

Группа клиентов

Входная очередь

Выходная очередь

**Cloud система реального времени**

Клиент

Клиент

\*\*\*

SSA

SSA

Канал обслуживания

Планировщик

Канал обслуживания

\*\*\*

Клиент

SSA

Клиент

SSA

SSA

SSA

Рисунок 6 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.

**6. 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.

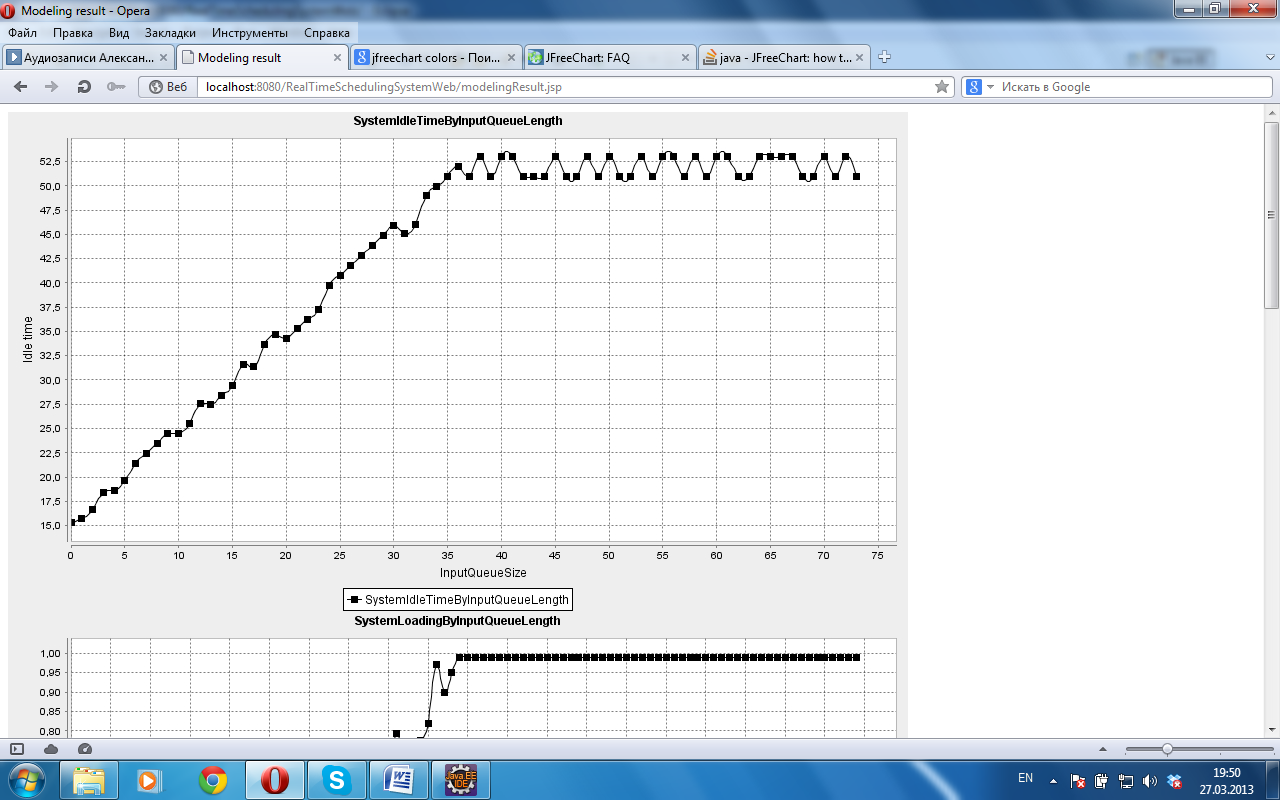


Рисунок 7 График зависимости ожидания завершения выполнения задачи от длинны входной очереди

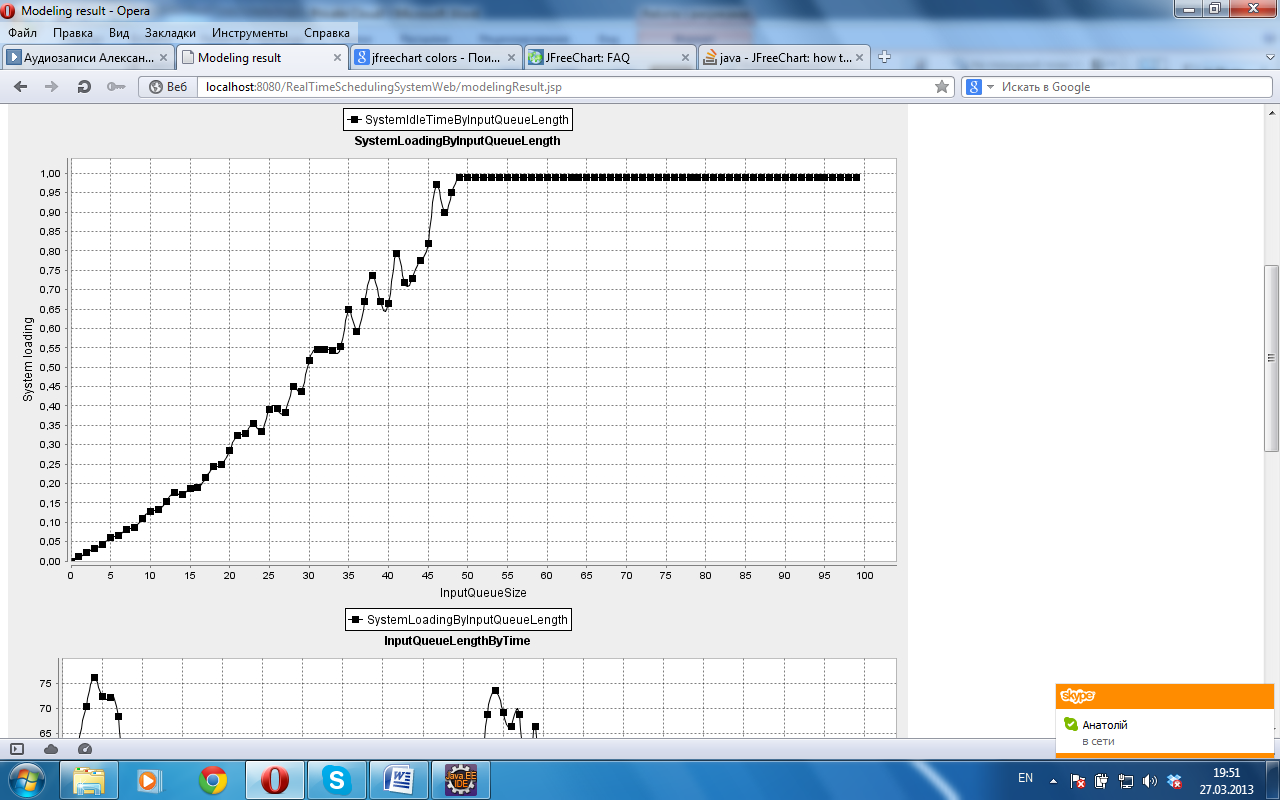


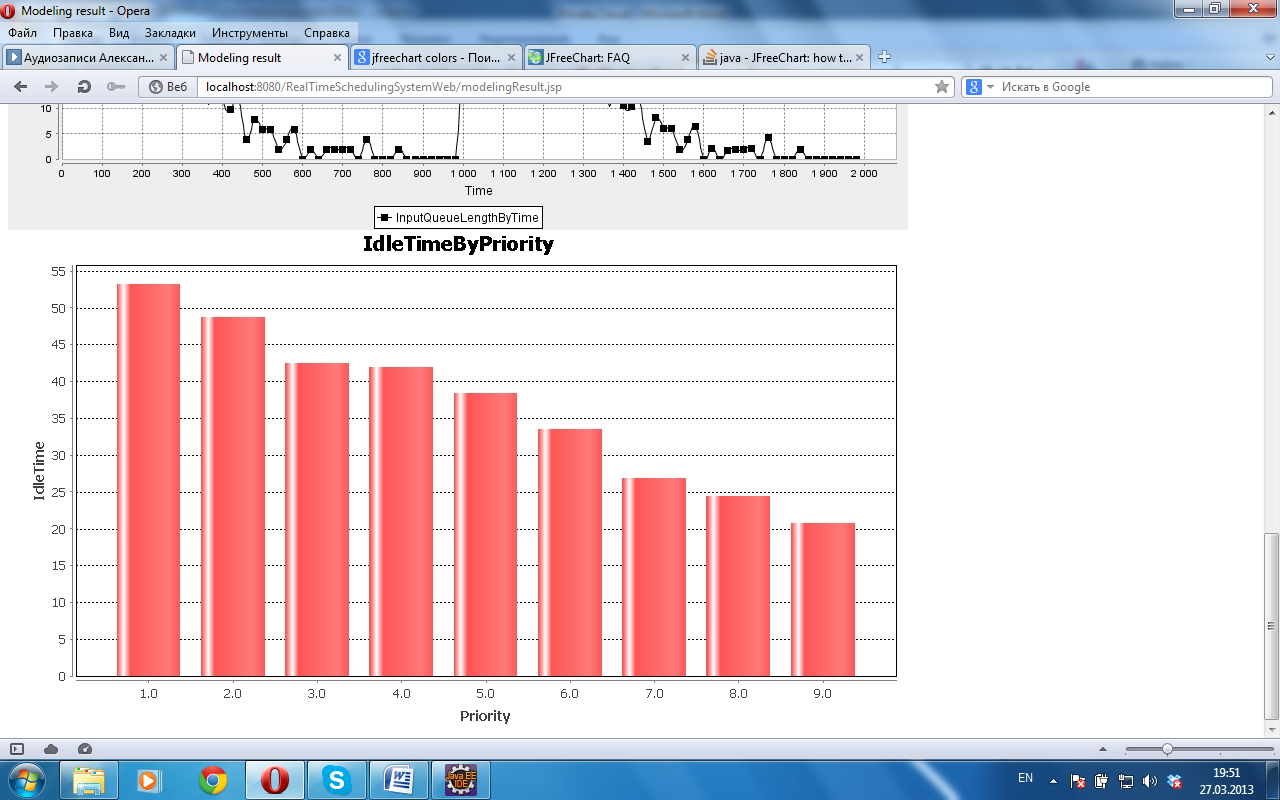
Рисунок 8 График зависимости загрузки системы от длинны входной очереди  


Рисунок 9 График зависимости времени ожидания задачи от уровня приоритета задачи

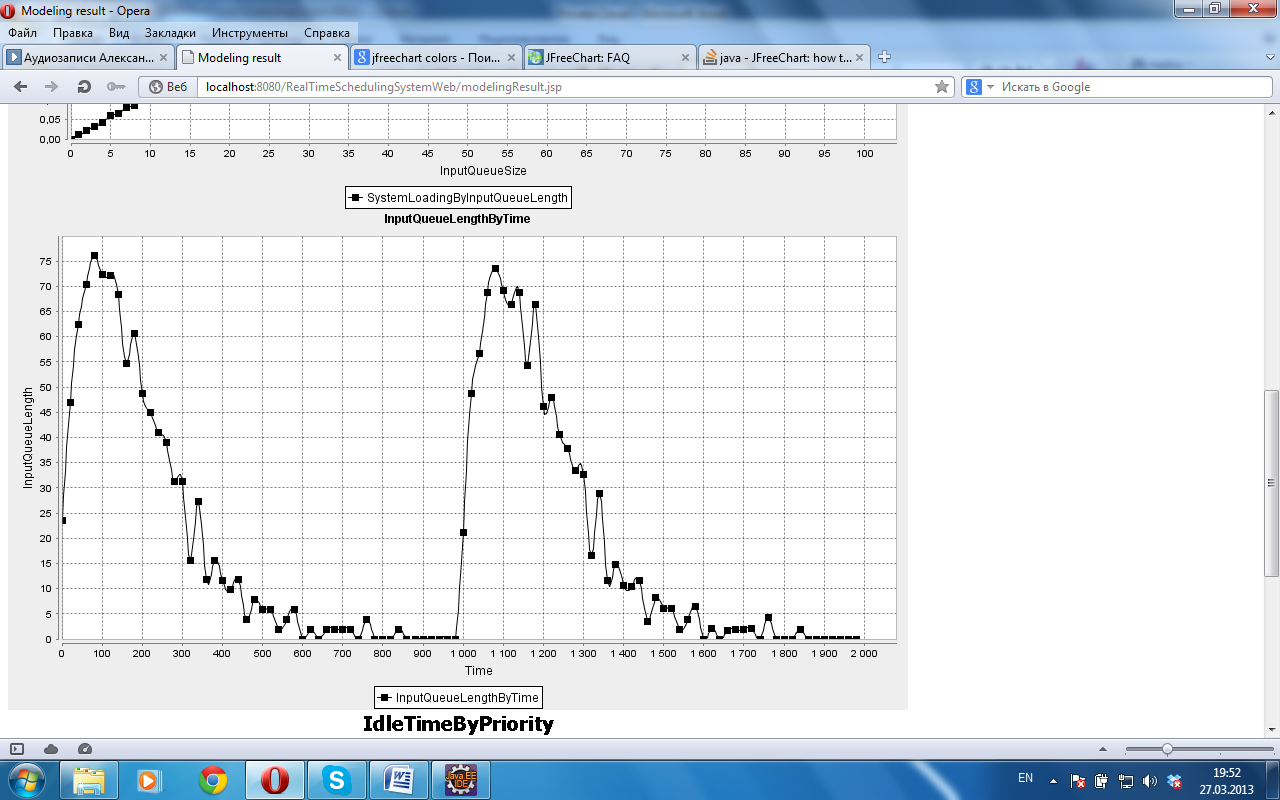


Рисунок 10 График зависимости длины входной очереди от времени

Как видно из рисунка 7, 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.

На рисунке 8 видно, что при увеличении количества задач во входной очереди, загрузка системы растет и когда наступает 100 процентная загрузка, задачи в систему не поступают.

Задачи с более высоким приоритетом проводят времени в ожидании меньше чем задачи с низким приоритетом, рисунок 9.

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.

**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