УДК 519.217+332.8
DOI 10.34822/1999-7604-2020-1-6-12

СИСТЕМНЫЙ АНАЛИЗ, МЕТОДЫ И АЛГОРИТМЫ ПРИНЯТИЯ РЕШЕНИЙ ПО ПОВЫШЕНИЮ ЭНЕРГОЭФФЕКТИВНОСТИ НАСЕЛЕННОГО ПУНКТА

И. В. Кутуев [1]✉, Д. А. Федоров [2], К. И. Бушмелева [2]
[1] Управление информационных технологий и связи города Сургута, Сургут, Россия
[2] Сургутский государственный университет, Сургут, Россия
✉E-mail: kutuevivan@gmail.com

Города сегодня сталкиваются с множеством проблем, среди которых ключевой является проблема энергоэффективности. Данной теме посвящено множество работ как в нашей стране, так и за рубежом. Одним из способов повышения энергоэффективности умного города является управление энергопотреблением в умных домах. В работе дан анализ двух способов повышения энергоэффективности умных городов с помощью системы обучения агентов и интернета вещей. Произведен сравнительный анализ предложенных методов по критерию «Степень снижения потребления энергии».

Ключевые слова: умный город, интернет вещей, повышение энергоэффективности, Марковский процесс принятия решения, многоагентные системы.

SYSTEM ANALYSIS, METHODS AND ALGORITHMS FOR DECISION MAKING TO INCREASE THE ENERGY EFFICIENCY OF SETTLEMENT

I. V. Kutuev [1]✉, D. A. Fedorov [2], K. I. Bushmeleva [2]
[1] Department of Information Technology
and Communication of Surgut City, Surgut, Russia
[2] Surgut State University, Surgut, Russia
✉E-mail: kutuevivan@gmail.com

Cities today face many challenges, among which energy efficiency is a key requirement. This problem has been the subject of many works both in Russia and abroad. One way to improve the energy efficiency of a Smart city is to manage energy consumption in smart homes. This paper provides an analysis of two ways to improve the energy efficiency of Smart cities using agent training systems and the Internet of things. A comparative analysis of the proposed methods in terms of "The degree of energy consumption reduction" is made.

Keywords: Smart city, Internet of things, energy efficiency, Markov decision process, multi-agent systems.

В настоящее время одной из наиболее перспективных областей развития информационных технологий является так называемый интернет вещей (далее – IoT). Одно из наиболее популярных направлений развития IoT – концепция «Умный город» (далее – SC).

Согласно концепции SC должен быть [1]:

1) технологичным;
2) интеллектуальным;
3) экологичным;
4) безопасным;
5) энергоемким;
6) открывающим широкие возможности и обеспечивающим максимально комфортную жизнедеятельность.

Разработкой стандартов SC в Российской Федерации занимается Национальный центр информатизации. Также в ближайшее время в России будет сформирован технический комитет по стандартизации «Кибер-физические системы».

Кратко умный город можно определить как город, который использует информационные и коммуникационные технологии (далее – ИКТ), такие как интеллектуальные датчики, когнитивное обучение и контекстная осведомленность, чтобы сделать жизнь более комфортной, эффективной и устойчивой. Сегодня города сталкиваются с разнообразными проблемами, включая экологическую устойчивость, низкоуглеродные решения и предоставление более качественных услуг своим гражданам. Учитывая эти тенденции, очень важно понять, как ИКТ могут помочь сделать города более устойчивыми.

Энергетические системы требуют равновесия между выработкой электроэнергии и спросом, поэтому необходимо ввести такое понятие, как управление спросом на электроэнергию, или Demand Response (далее – DR). DR подразумевает снижение энергопотребления конечным потребителем при определенных экономических сигналах рынка электроэнергии с получением выручки за осуществление такого снижения потребления [2].

В данной статье анализируются способы повышения энергоэффективности [3] умных городов с помощью системы обучения агентов и интернета вещей.

Уменьшение энергопотребления с помощью системы обучения агентов. «Агент» как развитие известного понятия «объект» является, по определению Международной ассоциации по лингвистике FIRA, «сущностью, которая находится в некоторой среде, интерпретирует ее и исполняет команды, воздействующие на среду» (октябрь 1996 г., Токио). Агент – это программный модуль, способный выполнять заданные ему функции некоторого живого или кибернетического организма в зависимости от функций другого агента и воздействий активной среды [4].

В статье [5] была сформулирована проблема сокращения пиковых значений DR на основе системы обучения агентов. Многие авторы пытались решить эту проблему, используя несколько инструментов, таких как прогнозное управление моделью, оптимизация роя частиц, итеративное динамическое программирование и градиентные методы. Тем не менее эти модели являются вероятностными и не представляют собой обучение взаимодействию с окружающей средой.

В работе [5] предлагается система управления энергопотреблением, названная «Домашнее управление энергией как услуга» (далее – HEMaaS), которая обеспечивает интеллектуальные решения, взаимодействует с окружающей средой, масштабируется и удобна для пользователя. Интеллектуальные датчики, подключенные по Wi-Fi, с централизованным механизмом принятия решений могут определять условия пиковой нагрузки и использовать автоматическое переключение для отвода или уменьшения потребности в энергии в пиковый период, тем самым снижая потребление энергии.

Авторы [5] использовали типичную канадскую жилую квартиру для исследования эффективности предлагаемой услуги по управлению энергопотреблением дома. Основная цель HEMaaS состоит в том, чтобы сократить использование бытовой техники и посредством этого снизить пиковый спрос и общее потребление энергии. Для этого был предложен новый алгоритм обучения на основе нейронной сети для достижения целей. Классическая проблема *Q*-обучения в обучении с подкреплением была сформулирована как задача обучения с нейронным контролем и называется алгоритмом управления энергопотреблением на основе *Q*.

Для достижения повышения энергоэффективности среды умного дома (далее – HEM) был использован Марковский процесс принятия решения. Проблема формулировалась как набор дискретных состояний, где каждое состояние представляло бы собой показатель уровней мощности бытовой техники, а главный блок управления и контроля (далее – MCCU) выдавал бы команду для переключения этих состояний питания.

Система действует следующим образом: MCCU выдает команду для переключения состояний питания, далее моделируются энергетические состояния как процесс принятия решения, после чего выводится Марковское решение (Markov decision process, далее – MDP) об использовании армирования обучения.

Марковское решение – это стохастический процесс управления с дискретным временем, в котором результаты получаются частично с помощью комбинации случайных событий и частично с помощью процесса принятия решений. На каждом временном шаге MDP модели-

руется как последовательность конечных состояний $s_i \in S$, действие агента $a_i \in A$, которые оцениваются на основе случайного процесса, чтобы привести агента в другое состояние.

Армирование обучения (далее – RL) является областью машинного обучения и касается того, как программные агенты должны принимать меры в окружающей среде так, чтобы максимизировать некоторое представление о совокупной награде.

Основное усиление моделируется как процесс принятия решений Маркова [6]:

1. Набор окружающей среды и агентов состояний, S.
2. Набор действий агента, A.
3. $P_a(s, s`) = \Pr(s_{t+1} = s` | s_t = s, a_t = a)$ – вероятность того, что действие a в состоянии s, во время t, приведет к состоянию на момент asts`t + 1.
4. $R_a(s, s`)$ – немедленное вознаграждение (или ожидается немедленное вознаграждение), полученное после перехода из состояния в состояние, из-за действия $ss`a$.
5. Правила, которые описывают то, что наблюдает агент.

Q-обучение – это онлайн-алгоритм, который выполняет обучение с подкреплением. Алгоритм вычисляет качество пары «состояние – действие», которая обозначается как Q и инициализируется нулем в начале фазы обучения. На каждом этапе взаимодействия со средой агент наблюдает за окружением и принимает решение о действии по изменению состояния в зависимости от текущего состояния системы. Новое состояние дает агенту вознаграждение, которое указывает на значение перехода состояния. Агент сохраняет функцию значения $Q^{\pi}(s(t), a(t))$ в соответствии с выполненным действием, которое максимизирует долгосрочное вознаграждение.

Уравнение обновления *Q*-фактора со скидкой вознаграждения выглядит следующим образом (1):

$$Q^{T+1}(s(t+1), a(t)) = Q^{T}(s(t), a(t)) + a(s(t), a(t)) \left(R_1(t) + \gamma \times MAX\left(Q^{T}(s(t), a(t))\right) - Q^{T}(s(t), a(t))\right), \quad (1)$$

где $\alpha(s(t), a(t))$ – скорость обучения ($0 < \alpha < 1$), γ – коэффициент дисконтирования в диапазоне 0 и 1. Если он близок к 0, агент выбирает немедленное вознаграждение, иначе он выберет долгосрочное вознаграждение, $R(t)$ – вознаграждение.

Результатом работы стал алгоритм управления энергопотреблением на основе нейронной *Q*-системы (рис. 1).

Input : Define $Q^0 = \overline{q}^0_{s,a} = 0, s_k = Th$
Output : $\pi(s)$

1 $ITER \leftarrow 200$
2 $T^k \leftarrow empty\ set$
3 $\theta \leftarrow randomweight$
4 **while** $(||Q(s+1,a) - Q(s,a)|| < 10^{-4})$ **do**
5 $\quad \overline{q}^i_{s,a} = r(t) + \gamma \cdot MAX\, \overline{Q}^{i-1}(s(t+1), a(t))$;
6 $\quad$ **for** 1:*ITER* **do**
7 $\quad\quad T^k \leftarrow T^{k-1} \cup (s, a; \overline{q}^{i+1}_{s,a})$;
8 $\quad\quad \delta \leftarrow \hat{Q}(s(t+1), a(t)) - \overline{Q}^{i}(s(t+1), a(t))$;
9 $\quad\quad \phi_k(s,a) \leftarrow exp^{-\frac{||s - \overline{s}_k||^2}{2 * \sigma_k^2}}$;
10 $\quad\quad \theta \leftarrow \theta + \alpha\delta\phi(s,a)$;
11 $\quad$ **end**
12 $\quad P(s,a) \leftarrow \frac{e^{ExplorationCount} \cdot age \cdot Q(s,a)}{\Sigma(e^{ExplorationCount} \cdot age \cdot Q(s,a))}$
13 **end**
14 return $\pi(s)$

Рис. 1. Алгоритм управления энергопотреблением [5]

Система состоит из Raspberry-Pi3 (далее – RP) с запущенным алгоритмом, написанным на языке Python, и брокера Mosquito, использованного как связующее звено между датчиками и RP.

Алгоритм работает в три этапа: разведка, обучение и применение. На этапе разведки собирает статистические данные о спросе, основанные на разных временах года. Данные зимнего месяца были выбраны в данном приложении.

Этап разведки состоит из следующих шагов:

Шаг 0 (входы): установите *Q*-факторы на некоторые произвольные значения (например, 0).

Шаг 1: для каждого состояния s определен набор допустимых действий a, и случайное действие $a \in A$ выбирается и применяется. После применения действия $a(t)$ в состояние $s(t)$ достигается следующее состояние – $s(t+1)$ и вычисляется непосредственное вознаграждение $R(t)$ из алгоритма вычисления Матрицы вознаграждений (R) [5].

Шаг 2: множество $(s(t), a(t), R(t), s(t+1))$ вставляется из окружающей среды в качестве нового образца F. При повторении процесса найдены достаточные образцы для обучения алгоритму.

Этап фазы обучения состоит из следующих шагов:

Шаг 1: обучение инициализируется $Q^0 = \overline{Q}_{s,a}^0 = 0$ и пытается найти аппроксиматор функции $\hat{Q}^i$.

Шаг 2: аналогично процессу *Q*-update добавить соответствующий набор шаблонов T^k к набору $(s, a;\ \overline{q}_{s,a}^{i+1})$.

Шаг 3: поскольку наши исторические данные являются проблемой подбора кривой, нейронная сеть с радиальной базисной функцией (RBFNN) выбрана для аппроксимации функции Q (s, a).

Шаг 4: функция признака ϕ: *S x A* отображает каждую пару состояния-действия в вектор значений признаков, где S – состояния и A – набор допустимых действий.

Шаг 5: θ-вектор весов, определяющий вклад каждого признака во всех парах состояния-действия. Вес обновляется на каждой итерации. Обучение проводится за 200 итераций.

Этап фазы исполнения состоит из следующих шагов:

Шаг 1: текущие данные определяют состояние системы.

Шаг 2: «жадная» политика используется для нахождения политики $\pi(s)$, как в уравнении (3):

$$\pi(s) = arg\ \max_{n \epsilon a} Q(s, a), \tag{2}$$

где s – состояния;

a – набор допустимых действий.

Шаг 3: в результате изучения большего количества эпизодов эксплуатация имеет больше смысла, потому что с опытом агент может быть более уверен в том, что он знает.

Шаг 4: остановка критерия с абсолютной ошибкой.

$$\| \left(Q^T(s+1, a)\right) - Q^T(s, a)\| < 10^{-4}, \tag{3}$$

где s – состояния;

a – набор допустимых действий;

Q – функция обучения.

После проведения экспериментов было выявлено, что авторам [5] удалось снизить пиковую нагрузку на 10 %, принимая во внимание, что другие уровни снижения пиковой мощности вызывают больше дискомфорта для пользователей.

Уменьшение энергопотребления с помощью IoT. Еще один способ снижения энергопотребления умных домов был описан Smruti R. Sarangi [7]. Автором была предложена архитектура системы, в которой каждый датчик соединен с одним шлюзом (рис. 2).

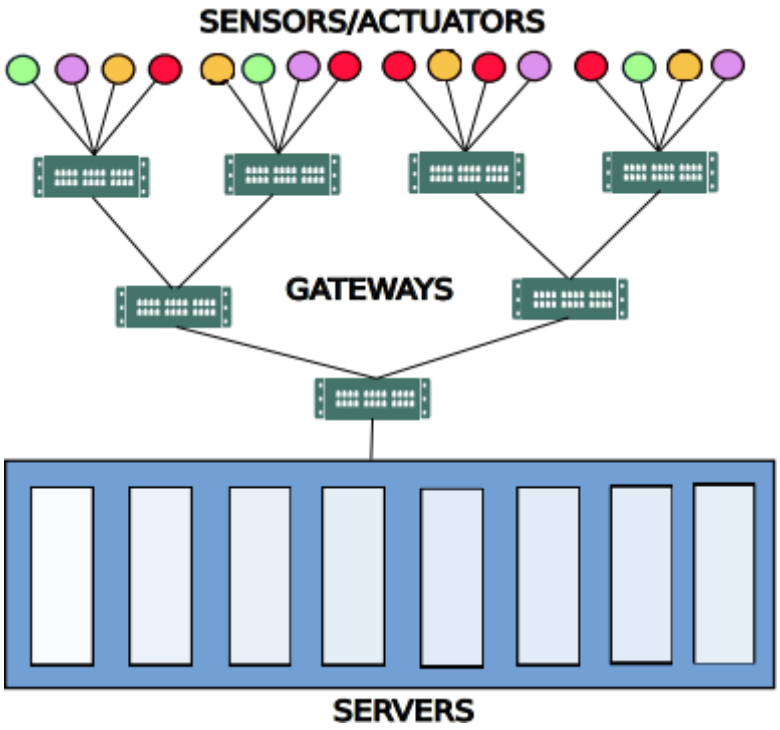

Рис. 2. Архитектура IoT системы [7]

Рассматривался поток задач ($t_1, t_2, \dots, t_i$), генерируемый датчиками. Задача t_i, поступающая в узел может быть сформулирована как кортеж (g_i, d_i, l_i, c_i), где: g_i – время генерации задания, d_i – дедлайн задания, l_i – суммарный сетевой трафик, связанный с заданием (в байтах), c_i – количество циклов выполнения, необходимое для задания.

Потребление энергии на узлах определяется процессором и памятью. Энергия, потребляемая процессором, состоит из двух компонентов: динамической энергии и статической энергии, которая также известна как энергия утечки [8]. Так как динамический компонент является основным источником потребления энергии систем и колебания температуры ограничены, модель потребления энергии ориентирована в основном на динамическое потребление энергии и предполагается, что статическая энергия постоянная.

На основании вышеизложенного было предложено два алгоритма для энергоэффективного планирования задач в IoT. В обоих алгоритмах каждый узел обработки использует t_{est}, расчетное время, которое потребуется для достижения цели тогда, когда он покидает текущий узел, чтобы вычислить t_{erm} (максимальное доступное время, за которое узел должен выполнить текущую задачу). Это делается с учетом дедлайна. Впоследствии многоядерный узел выполняет масштабирование на основе DVFS (масштабирование или уменьшение частоты некоторого ядра).

Результатом работы стали два алгоритма: глобальный и локальный.

В глобальном алгоритме (рис. 3) сохраняются центральный сервер с высокой пропускной способностью и компьютерная система (CS), для которой доступны все узлы, так что они могут связываться с CS и наоборот. Этот алгоритм более полезен в системе, имеющей много шлюзов, которые обладают некоторой формой прямого подключения к Интернету. Предполагается, что все общение с CS имеет самый высокий приоритет.

Algorithm 3 t_{est} calculations at the Central Server with the *Global* Algorithm

```
1:  procedure GlobalAlgoAtCS
2:      receive(request);
3:      n ← request.requestingNode;
4:      t_avg ← request.message;
5:      if request.reqCode = get_t_est_value then
6:          curnode, prevnode ← n
7:          while actuator node is not reached do
8:              curnode ← n.nextNode
9:              add t_avg for tuple (curnode, prevnode) to t_send
10:             add propagation delay for link between curnode and
    prevnode
11:             prevnode ← curnode
12:         end while
13:         res.message = t_send;
14:         send(res, n);
15:     end if
16:     if request.reqCode = send_t_avg_value then
17:         Update the t_avg for all the child nodes of node n
18:     end if
19: end procedure
```

Рис. 3. Вычисления в рамках глобального алгоритма [7]

В локальном алгоритме (рис. 4) каждый узел обновляет свое t_{est} значение с использованием информации, совмещенной с задачей, которую он получает от соседних узлов. Каждый узел вычисляет среднее время, которое тратит задача в узле, путем усреднения времени интервала $t_d - t_a$ (где t_a – время прибытия задачи в узел и т. д., t_d – время отправления задачи из узла) для всех задач, выполняющихся на узле. Этот интервал времени включает в себя:

1. Время ожидания задачи во входной очереди узла.
2. EDF-очередь ядра, которому она назначена.
3. Время выполнения задания на назначенном ядре.

Кроме того, каждый узел поддерживает информацию о предполагаемом времени (t_{est}) для каждого из соседних с ним узлов.

Algorithm 4 t_{est} calculations in the *Local* Algorithm

```
1:  procedure Local
2:      receive(request);
3:      task ← request.task;
4:      (info_recv, source) ← (request.message, request.source);
5:      t_est for source ← (t_est for source + info_recv ) / 2;
6:      info_send ← t_avg + t_est for source;
7:      t_est for self ← t_est for next node
8:      executeTask(task, t_est);
9:      (req.task, req.message) = (task, info_send);
10:     send(req, self.nextNode);
11: end procedure
```

Рис. 4. Вычисления в рамках локального алгоритма [7]

Автором [8] был разработан симулятор IoT на языке Java. Датчики, находящиеся в сети, были представлены с помощью бесплатного симулятора сетей NS3. Для симуляции центра обработки данных использовался один из самых популярных облачных симуляторов с открытым исходным кодом CloudSim.

Оба алгоритма предполагают обмен информацией между узлами, чтобы дать оценку оставшегося времени, необходимого для выполнения задачи, и выполнить масштабирование по частоте напряжения. Глобальный алгоритм использует выделенный центральный сервер,

что позволяет вычислить лучшую конфигурацию для всей сети. Локальный алгоритм получает эту информацию из совмещенных данных, которые поступают наряду с регулярными сообщениями от своих соседей.

После построения алгоритма была проведена симуляция работы, результаты которой показали, что обе предложенные схемы достигают значительного снижения энергопотребления (около 40 %) по сравнению с простыми DVFS, основанными на методах с небольшим или никаким ухудшением в производительности или по пропущенным срокам. При этом глобальный алгоритм работает лучше в случае больших сетей. Для задач со свободными сроками может быть предложен локальный алгоритм.

Заключение. В результате рассмотрения двух методов повышения энергоэффективности SC были получены результаты снижения энергии, указанные в таблице.

Таблица

Критерии эффективности методов

Наименование метода	Снижение потребления энергии
Обучение агентов	10 %
Планирование задач в IoT	40 %

Примечание: составлено авторами.

Как видно из таблицы, алгоритм планирования задач является более эффективным. В рамках исследования рассмотрен вопрос о повышении энергоэффективности SC. На основании вышеизложенного было принято решение о возможности использования метода планирования задач в дальнейшем исследовании.

Литература

1. Умный город: концепция, стандартизация и реализация смарт-сити. URL: http://1234g.ru/novosti/smart-city (дата обращения: 19.11.2019).

2. Технология ценозависимого потребления. URL: https://so-ups.ru/?id=dr (дата обращения: 19.11.2019).

3. Кутуев И. В., Бушмелева К. И. Анализ информационных систем используемых для расчета искусственного освещения // Инновационные, информационные и коммуникационные технологии : сб. трудов XV Междунар. науч.-практ. конф. / под. ред. С. У. Увайсова. М. : Ассоциация выпускников и сотрудников ВВИА им. проф. Жуковского, 2019. С. 71–75.

4. Ивашкин Ю. А. Мультиагентное моделирование в имитационной системе Simplex3. М. : Лаборатория знаний, 2016. 350 с.

5. Mahapatra C., Moharana A. K., Leung V. C. M. Energy Management in Smart Cities Based on Internet of Things: Peak Demand Reduction and Energy Savings // Sensors (Basel). 2017. No. 17 (12). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5751629/ (дата обращения: 22.11.2019).

6. Армирование обучения. Reinforcement learning. URL: https://ru.qwe.wiki/wiki/Reinforcement_learning (дата обращения: 23.03.2020).

7. Sarangi S. R., Goel S., Singh B. Energy Efficient Scheduling in IoT Networks // SAC 2018: Symposium on Applied Computing, April 9–13, 2018, Pau, France. ACM, NewYork, NY, USA, 8 p. https://doi.org/10.1145/3167132.3167213.

8. Yan L., Luo J., Jha N. K. Joint Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2005. Vol. 24, No. 7. P. 1030–1041.