MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 4, 2020
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

Knowledge-based Algorithms for BDI-agents

N. V. Shilov[1], N. O. Garanina[2]

DOI: 10.18255/1818-1015-2020-4-442-453

[1]Innopolis University, 1 Universitetskaya, Innopolis, 420500, Russia.
[2]A.P. Ershov Institute of Informatics Systems (IIS), Siberian Branch of the Russian Academy of Sciences, 6 Acad. Lavrentjev ave., Novosibirsk 630090, Russia.

MSC2020: 93A16
Research article
Full text in Russian

Received November 20, 2020
After revision December 5, 2020
Accepted December 16, 2020

Multiagent algorithm is a knowledge-based distributed algorithm that solves some problems by means of cooperative work of agents. From an individual agent's perspective, a multiagent algorithm is a reactive and proactive knowledge/believe-based rational algorithm aimed to achieve an agent's own desires. In the paper we study a couple of knowledge-based multiagent algorithms. One particular algorithm is for a system consisting of agents that arrive one by one (in a non-deterministic order) to a resource center to rent (for a while) one of available desired resources. Available resources are passive, they form a cloud; each of the available resources is lent on demand if there is no race for this resource and returns to the cloud after use. Agents also form a cloud but leave the cloud immediately when they rent a desired resource. The problem is to design a knowledge-based multiagent algorithm, which allows each arriving agent eventually to rent some of desired resources (without race for these resources).

Keywords: multiagent systems; multiagent algorithms; BDI-agents; knowledge and belief

INFORMATION ABOUT THE AUTHORS

Nikolay Vyacheslavovich Shilov correspondence author	orcid.org/0000-0001-7515-9647. E-mail: shiloviis@mail.ru Ph.D. in Mathematics, head of laboratory.
Natalia Olegovna Garanina correspondence author	orcid.org/0000-0001-9734-3808. E-mail: garanina@iis.nsk.su Ph.D. in Mathematics, senior research fellow.

Funding: This research has been financially supported by the Ministry of Digital Development, Communications and Mass Media of the Russian Federation and Russian Venture Company (Agreement No. 004/20 dd. 20.03.2020, IGK 0000000007119P190002.

For citation: N. V. Shilov and N. O. Garanina, "Knowledge-based Algorithms for BDI-agents", *Modeling and analysis of information systems*, vol. 27, no. 4, pp. 442-453, 2020.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 4, 2020
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

Алгоритмы для BDI-агентов, основанные на знаниях

Н. В. Шилов[1], Н. О. Гаранина[2]

DOI: 10.18255/1818-1015-2020-4-442-453

[1]Университет Иннополис, Университетская, д.1, г. Иннополис, 420500 Россия.
[2]Институт систем информатики имени А. П. Ершова СО РАН, пр. Лаврентьева, д. 6, г. Новосибирск, 630090 Россия.

УДК 004.8
Научная статья
Полный текст на русском языке

Получена 20 ноября 2020 г.
После доработки 5 декабря 2020 г.
Принята к публикации 16 декабря 2020 г.

Мультиагентный алгоритм — это распределённый алгоритм, основанный на знаниях, который решает некоторую проблему посредством совместной работы агентов. BDI-агент — это агент, обладающий убеждениями (Belief), желаниями (Desire) и намерениями (Intention). С точки зрения такого агента, мультиагентный алгоритм — это алгоритм, основанный на его знаниях и убеждениях, с помощью которого достигается выполнение его желаний посредством последовательного осуществления намерений. Мы считаем также, что агенты реактивны, проактивны и рациональны. В этой статье мы предлагаем и изучаем два мультиагентных алгоритма, которые основаны на знаниях. В частности, мы предлагаем мультиагентный алгоритм для следующей задачи аренды ресурсов. Система состоит из агентов, которые прибывают один за другим в произвольном порядке в ресурсный центр, чтобы арендовать один из предоставляемых ресурсов. Предоставляемые ресурсы пассивны, они образуют облако. Если за ресурс нет конкуренции, то он предоставляется по запросу, и возвращается в облако после использования. Агенты также образуют облако, но когда арендуют нужный ресурс, то сразу же покидают ресурсный центр. Задача состоит в разработке мультиагентного алгоритма, основанного на знаниях, обладающего следующим свойством корректности: каждый прибывающий в ресурсный центр агент рано или поздно арендует какой-либо из запрашиваемых ресурсов без конкуренции за этот ресурс в данный момент.

Ключевые слова: мультиагентные системы; мультиагентные алгоритмы; BDI-агенты; знания и мнения

ИНФОРМАЦИЯ ОБ АВТОРАХ

Николай Вячеславович Шилов	orcid.org/0000-0001-7515-9647. E-mail: shiloviis@mail.ru
автор для корреспонденции	канд. физ.-мат. наук, нач. лаб.
Наталья Олеговна Гаранина	orcid.org/0000-0001-9734-3808. E-mail: garanina@iis.nsk.su
автор для корреспонденции	канд. физ.-мат. наук, с.н.с.

Финансирование: Исследование выполнено при финансовой поддержке Министерства цифрового развития, связи и массовых коммуникаций РФ и АО «Российская венчурная компания» (договор №004/20 от 20.03.2020, ИГК 0000000007119P190002).

Для цитирования: N. V. Shilov and N. O. Garanina, “Knowledge-based Algorithms for BDI-agents”, *Modeling and analysis of information systems*, vol. 27, no. 4, pp. 442-453, 2020.

Введение: основные понятия

Термины “мультиагентный” и “знание” относятся к нескольким (иногда не связанным) концепциям, подходам и парадигмам исследований в теоретическом программировании, искусственном интеллекте, программной инженерии, анализе данных, информационных технологиях и т.д. В противоположность этому, понятие “облако” обычно используется лишь в контексте информационных технологий и программной инженерии, а со стороны искусственного интеллекта ему уделено незначительное внимание. В этой статье мы изучаем основанные на знаниях алгоритмы для системы BDI-агентов, которые запрашивают ресурсы, образующие облако.

Распределённая система [1, 2] состоит из нескольких автономных отдельных “компьютеров” (программ с распределённой памятью), которые обмениваются данными через сеть. Коммуникация в распределённой системе считается *справедливой*, если каждый компьютер, которому необходимо взаимодействовать с каким-либо другим компьютером, рано или поздно будет с ним взаимодействовать. Для обеспечения справедливости, очевидно, требуется планировщик или специальный механизм коммуникаций.

Мультиагентная система — это распределённая система, состоящая из агентов. *BDI-агент* [3] (далее просто *агент*) — это автономный реактивный и проактивный объект (в объектно-ориентированном смысле), внутренние состояния которого могут быть охарактеризованы в терминах убеждений (мнений) (B), желаний (D), и намерений (I).

Убеждения агента представляют его идеи/мнения о себе и окружении, т.е. других агентах и сети; эти убеждения могут быть неверными, неполными и даже несогласованными. *Желания* агента представляют собой его долгосрочные цели, обязательства и задачи, которые также могут быть противоречивыми. *Намерения* агента характеризуют его краткосрочное планирование. *Логическое всеведение* агента означает, что агент сразу знает все логические следствия, вытекающие из его знаний.

Мы различаем убеждение и знание согласно Платону, который полагал, что *знание* — это истинное мнение, имеющее подтверждение [4, 5]. В настоящей статье мы считаем, что убеждение агента становится знанием, если существует формальное доказательство истинности этого убеждения. Мы предполагаем логическое всеведение агента, и поэтому для того, чтобы убеждение стало знанием, достаточно истинности этого убеждения.

Реактивность агента означает, что он может изменить свои убеждения после общения и взаимодействия с другими агентами. *Проактивность* агента означает, что он может изменить свои намерения после изменения/обновления своих убеждений, т. е. планировать своё поведение в ближайшем будущем на основе имеющихся данных.

Каждый агент *автономен*, то есть изменение его личных убеждений, желаний и намерений не может быть предписано каким-либо другим агентом. *Рациональный* агент имеет чёткие предпочтения и всегда из возможных действий выбирает то, которое приводит к наилучшему личному результату.

Мультиагентный алгоритм — это распределённый алгоритм (протокол распределённой системы), который решает некоторую задачу посредством совместной работы агентов в мультиагентной системе. Родственной парадигмой в экономической науке является дизайн механизмов [6].

Отказоустойчивость мультиагентного алгоритма — это способность правильно решить задачу, несмотря на возможный сбой сети и/или некорректное поведение отдельных агентов.

Алгоритм, основанный на знаниях [7], — это алгоритм, в котором агент предпринимает действие только тогда, когда “знает”, что действие можно/нужно выполнить. В нашем случае это означает, что действие с общим ресурсом осуществляется только тогда, когда агент знает, что доступ к этому ресурсу является безопасным, т.е. при отсутствии конкуренции (гонок) за этот ресурс.

Классический пример мультиагентного алгоритма, основанного на знаниях — решение задачи о разрезании торта [3]. Ещё один популярный пример мультиагентного алгоритма, основанного на знаниях — решение головоломки о чумазых ребятишках [7].

Далее в разделах 1 и 2 мы предложим и изучим два новых (насколько нам известно) мультиагентных алгоритма, основанных на знаниях. Второй из этих алгоритмов (см. раздел 2) решает задачу о доступе к ресурсу в *облаке ресурсов*. Поэтому прежде, чем перейти к обсуждению алгоритмов, опишем, что такое *облачные вычисления*.

Согласно [8], "облачные вычисления — это модель для обеспечения повсеместного, удобного сетевого доступа по требованию к общему набору конфигурируемых вычислительных ресурсов (например, сетей, серверов, хранилищ, приложений и услуг), которые могут быть быстро заняты и освобождены с минимальными усилиями руководства или взаимодействия с поставщиком услуг". Определение включает пять основных характеристик, три модели обслуживания и четыре модели развертывания. Перечислим эти характеристики и модели.

- Характеристики: самообслуживание по запросу, широкий доступ к сети, пул ресурсов, масштабируемость (rapid elasticity), измеряемый сервис.
- Модели обслуживания: программное обеспечение как услуга (Software as a Service, SaaS), платформа как услуга (Platform as a Service, PaaS), инфраструктура как услуга (Infrastructure as a Service, IaaS).

1. Задача о хосте и P2P-серверах

В этом разделе мы рассмотрим задачу и мультиагентный алгоритм (протокол), которая до некоторой степени является модификацией задачи о чумазых ребятишках. Архитектура сети для этой задачи представлена на рисунке 1.

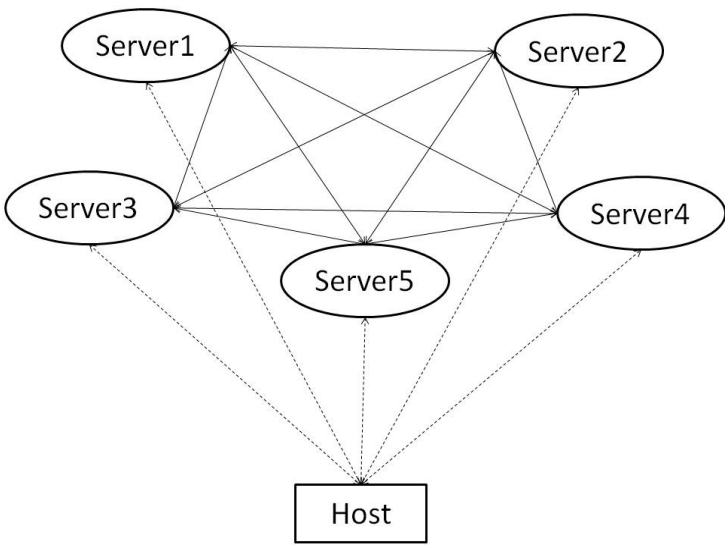

Fig. 1. Network for the problem of host and P2P-servers

Рис. 1. Архитектура сети для задачи о хосте и P2P-серверах

Постановка задачи

Система состоит из клиентского компьютера-хоста и n серверов, соединённых сетью. Хост соединяется с каждым сервером по выделенному каналу, и, кроме того, серверы связаны друг с другом в одноранговую сеть. Все соединения надёжны и мгновенны, хост подключён к каждому серверу напрямую, а каждый сервер имеет коммутатор, через который он подключается к одноранговой

сети. Таким образом, каждый сервер имеет свой индивидуальный коммутатор с $(n-1)$ сокетами для отдельных каналов к каждому из остальных $(n-1)$ серверов. По сети могут пересылаться оригинальные *сообщения* и *подтверждения* о получении.

Исправный коммутатор предоставляет своему владельцу-серверу i ($1 \le i \le n$), следующие два сервиса. Пусть j — это номер другого сервера: $1 \le j \le n$ и $j \ne i$.

1. Если владелец отправляет исходящее сообщение какому-либо серверу j, то коммутатор выбирает подходящий канал, пересылает сообщение, ожидает подтверждения от получателя j и затем уведомляет владельца о подтверждении.
2. Если коммутатор получает входящее сообщение или подтверждение от какого-либо сервера j, то он передаёт сообщение своему серверу-владельцу и, если было получено сообщение, то он высылает подтверждение отправителю сообщения серверу j.

Мы рассматриваем случай, когда коммутаторы ненадёжны: коммутатор всегда может предоставить первый сервис по отправке сообщений, но может раз и навсегда отказать в предоставлении второго сервиса для всех входящих сообщений и подтверждений со всех других серверов, т.е. перестать передавать полученное сообщение/подтверждение своему серверу-владельцу и посылать подтверждение серверу, отправившему сообщение. Напомним, связь с хостом осуществляется по специальному каналу без коммутатора, поэтому сбой коммутатора не может прервать связь с хостом.

У серверов нет аппаратных средств для проверки исправности своих коммутаторов. Кроме того, неясно, как сервер может проверять работоспособность своего коммутатора логически на основе анализа входящего трафика, так как следующие случаи отсутствия коммуникации неразличимы с поломкой коммутатора:

- если нет входящих сообщений, то, возможно, другим серверам нечего отправлять;
- если нет подтверждений для исходящих сообщений, то, возможно, вышли из строя коммутаторы всех остальных серверов.

Предположим, что в какой-то момент некоторые коммутаторы неисправны в описанном выше смысле, и множество таких коммутаторов не меняется.

Задача: разработать основанный на знаниях мультиагентный алгоритм для системы хоста и серверов, гарантирующий следующие свойства прогресса и безопасности [9].

1. Прогресс: каждый сервер рано или поздно узнает об исправности или неисправности своего коммутатора.
2. Безопасность: каждый сервер сообщит хосту истинное состояние (исправен или неисправен) своего коммутатора.

Решение, основанное на синхронизации

Введём следующие обозначения для временных интервалов:

- *SCR* (Server Communication Round) — это время, которое достаточно каждому серверу для отправки сообщений другим серверам (не более чем по одному сообщению), для ожидания получения подтверждений и для выполнения внутренней обработки данных (deliberation).
- *HSR* (Host-Server Round) — это время, которое достаточно хосту для объявления всем серверам (рассылки сообщений), для выполнения внутренней обработки данных серверами и хостом, и для приёма отдельных сообщений от серверов (не более чем по одному сообщению).

Задача может быть решена с помощью следующего протокола, аналогичного алгоритму задачи о чумазых ребятишках [7]. Протокол состоит из двух частей: клиентской части для исполнения на хосте и серверной части для исполнения на каждом сервере.

Клиентская часть

1. Сначала создать переменную для хранения списка отчётов от серверов и инициализировать ее пустым списком, а затем дождаться окончания временного интервала $HSR + SCR$.

2. Пока пуст список отчётов от серверов повторять следующий цикл:
   сначала разослать новый запрос отчётов всем серверам, а затем собрать в переменной для хранения отчётов все полученные отчёты с серверов, которые поступят до истечения очередного временного интервала HSR.
3. Пополнить переменную для хранения отчётов всеми отчётами, которые поступят от серверов до истечения следующего временного интервала HSR, а затем остановиться (завершить работу алгоритма).

Серверная часть

1. Отправить всем серверам сообщение, а затем подсчитать в своей приватной переменной FS количество коммутаторов других серверов, от которых не пришло подтверждения до истечения временного интервала $HSR + SRT$.
2. Пока $FS > 0$ повторить следующий цикл:
   если до истечения временного интервала HSR получен новый запрос об отчёте от хоста,
   то уменьшить значение своей приватной переменной FS и дождаться окончания текущего временного интервала HSR
   иначе — отправить отчёт на хост о неисправности собственного коммутатора, выйти из цикла (break) и остановиться (завершить работу алгоритма).
3. Когда $FS = 0$ и в течение следующего интервала времени HSR поступает новый запрос от хоста о предоставлении отчёта, то отправить отчёт на хост об исправности собственного коммутатора и остановиться (завершить работу алгоритма).

Корректность и анализ решения

Утверждение 1. *Предположим, что в системе все агенты (хост и серверы) начинают работать одновременно (синхронно) и в системе есть хотя бы два исправных коммутатора. Тогда мультиагентный алгоритм, основанный на синхронизации, гарантирует свойства прогресса и безопасности.*

Идея доказательства.
Мы считаем, что хост и все серверы являются BDI-агентами. У каждого агента есть желание узнать, правильно ли работает его коммутатор или вышел из строя. Пусть m — количество неисправных коммутаторов.

Прежде всего отметим, что временные интервалы играют роль барьеров синхронизации, т.е. первый шаг хоста завершается одновременно с первым шагом каждого сервера, и все итерации цикла 2 хоста и цикла 2 каждого сервера также завершаются одновременно.

После начала работы каждый сервер i ($1 \le i \le n$) подсчитывает в своей приватной переменной FS количество m_i других серверов, от которых не пришло подтверждение; с этого момента сервер i считает, что m_i — это количество сломанных коммутаторов. Заметим, что для любого $1 \le i \le n$, если коммутатор сервера i исправен, то $m_i = m \le (n - 2)$, а если неисправен, то $m_i = (n - 1)$. (Именно здесь используется то, что как минимум два коммутатора исправны: иначе у всех серверов значение приватной переменной FS будет $(n - 1)$.)

Поэтому хост и все серверы (как с исправными, так и с неисправными коммутаторами) синхронно исполняют первые m итераций своих циклов 2. Так как во время этих первых m итераций хост не получал ни одного отчёта, то он продолжает итерации этого цикла и отправляет ещё один запрос всем серверам. Но у серверов с исправными коммутаторами в это время значение приватной переменной FS равно 0, следовательно они уже вышли из своего цикла 2 и в ответ на очередной запрос хоста отправляют отчёт об исправности своих коммутаторов (шаг 3). Хост после получения отчётов от (как минимум 2) серверов с исправными коммутаторами выходит из своего цикла 2 и не посылает запросов серверам на следующем интервале времени HSR. В свою очередь, серверы с неисправными коммутаторами во время этого временного интервала все ещё выполняют свой

цикл 2, значения их приватных переменных FS все ещё больше 0, и так как они не получают запрос от хоста, то отправляют отчёт о неисправности своих коммутаторов.

Оценим временную сложность алгоритма, основанного на синхронизации. Запросы отчётов будут отправлены хостом m раз; поскольку время между последовательными запросами составляет HSR, тогда цикл 2 хоста требует время $m \times HSR$. Принимая во внимание шаги 1 и 3 хоста, общее время равно $(m+2) \times HSR + SRT$. Таким образом, сложность линейно зависит от общего числа неисправных коммутаторов.

Отметим, что основная проблема (которую мы пока не знаем как решить) алгоритма, основанного на синхронизации, — отказоустойчивость: если какой-либо из серверов не будет исполнять предписанный алгоритм, результат работы будет неверным.

Отметим также, что синхронизация является существенным элементом нашего решения как и в задаче о чумазых ребятишках: в задаче о чумазых ребятишках роль синхронизации выполняют запросы отца к детям. К сожалению, нам не известны какие-либо решения задачи без синхронизации, сохраняющие автономность и приватность агентов.

2. Задача о ресурсном центре

Постановка задачи

Ресурсный центр состоит из

- "облака" из $m > 0$ разных неделимых единичных ресурсов для аренды,
- "облака" из $n > 0$ агентов, которым нужны единичные ресурсы в аренду, и
- монитора.

Монитор ресурсного центра

- обеспечивает агентов данными, которые агенты могут использовать для разрешения конфликтов (например, мгновенно регистрирует участников и маркирует их отметками времени по прибытии в центр, т. е. работает как электронная очередь);
- мгновенно назначает (сдаёт в аренду) доступный ресурс агенту, если он в данный момент является *единственным* агентом, запрашивающим этот ресурс, но не назначает ресурс никому, если в данный момент два или более агента запрашивают этот ресурс;
- в случае конкуренции за ресурс регистрация новых участников прекращается до тех пор, пока ресурс не будет сдан в аренду;
- ведёт реестр ресурсов доступных для аренды (в частности, освобождённых после аренды).

Агенты прибывают в ресурсный центр в произвольном асинхронном порядке (и никогда одновременно). Каждый агент намерен арендовать на время желаемый и доступный ресурс, а затем когда-нибудь возвратить его в центр. У каждого агента есть собственный индивидуальный приватный (не подлежащий публичному объявлению) список желаемых ресурсов. Агенты общаются друг с другом через надёжную P2P-сеть с мгновенным соединением.

Индивидуальные списки желаемых ресурсов разных агентов могут пересекаться. Каждый список отсортирован в соответствии с индивидуальными приоритетами агента. Любой элемент в списке может удовлетворить текущие потребности агента, т. о. агенту нужны не все элементы, а только один. Список желаний может измениться, если агент в данный момент не зарегистрирован монитором ресурсного центра.

Наш центр ресурсов, как облако, абстрагируется от моделей обслуживания, но в значительной степени обладает описанными в разделе облачными характеристиками.

- Самообслуживание по запросу: агенты могут пользоваться ресурсами, самостоятельно разрешая возникающие конфликты.
- Широкий доступ к сети: в текущей постановке задачи эта характеристика несущественна.
- Пул ресурсов: агенты выбирают из множества доступных ресурсов, перераспределяя выбор в случае необходимости.

- Масштабируемость: алгоритм функционирования агентов в центре не зависит от их количества.
- Измеряемый сервис: в текущей постановке задачи эта характеристика несущественна.

Задача

Разработать основанный на знаниях мультиагентный алгоритм, который гарантирует, что каждый агент, прибывший в центр ресурсов, рано или поздно получит какой-либо ресурс из своего списка желаний.

Назовём задачу и мультиагентный протокол её решения (алгоритм, представленный в следующем подразделе) *гонкой за ресурсы*.

Пример конфигурации задачи (т.е. мгновенный снимок в некоторый момент времени) приведён на рисунке 2. Поясним списки желаний агентов в этом примере:

- “книга на 2 часа < теннис на 1 час” в списке желаний первого агента означает, что он предпочитает арендовать книгу на 2 часа, чем играть в теннис 1 час.
- “книга на 1 час < футбол на 1 час *или* теннис на 1 час” в списке желаний второго агента означает, что он предпочитает (а) взять напрокат книгу на 1 час, чем взять футбольный мяч на 1 час, и (б) взять книгу на 1 час, чем взять теннисный набор на 1 час, а также (в) аренда футбольного мяча на 1 час и аренда теннисного набора на 1 час имеют одинаковый приоритет.

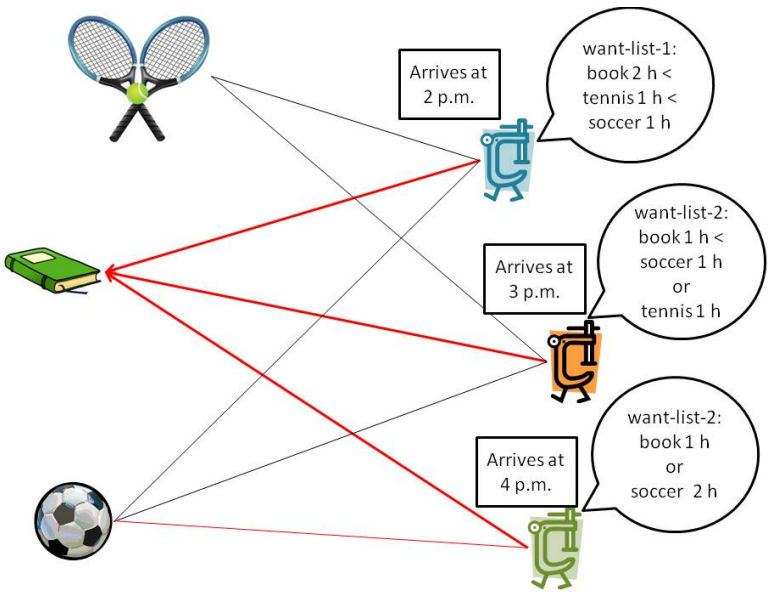

Fig. 2. Examples of agent requests

Рис. 2. Примеры запросов агентов

Нотация используемая в описании алгоритма

Алгоритм представляет собой пересмотренный мультиагентный алгоритм, основанный на знаниях из [10]. В обновлённом алгоритме мы различаем два случая: либо все агенты *мгновенные*, либо все агенты *реального времени*. Мгновенным агентам не нужно время для выполнения действий, в то время как каждое действие агентов реального времени выполняется с задержкой, и длительность этой задержки точно не известна. Мы обсудим разницу между мгновенными и агентами реального времени ниже.

Мы предполагаем, что каждый агент в любое время может запросить монитор о составе множества $POOL$ агентов, которые в настоящее время зарегистрированы. В каждый момент агент $i \in POOL$ устанавливает в качестве текущего намерения некоторый элемент ресурсов из своего списка же-

ланий $Desires_i \subseteq Resources$; в самом начале намерением является первый элемент этого списка. Конфликт (конкуренцию или гонку) между агентами, т.е. ситуацию, когда два агента выбирают один и тот же ресурс, можно выявить с помощью предиката $CONF : Agents \times Resources \times Agents \times Resources \longrightarrow Boolean$, так что $CONF(i, u_i, j, u_j) = true$ тогда и только тогда, когда $u_i = u_j$ ($i \neq j$, $i, j \in POOL$). Очевидно, что для всех агентов $i \neq j$, для всех элементов ресурсов u_i и u_j отношение конфликта симметрично: $CONF(i, u_i, j, u_j) \iff CONF(j, u_j, i, u_i)$. Мнения каждого агента представлены следующими приватными целочисленными счётчиками: счётчик $NumC$ для общего числа конфликтов используется как мгновенными агентами, так и агентами реального времени, а счётчик $CFree$ нужен только последним для подсчёта агентов без конфликтов.

Поясним подробнее разницу между мгновенными агентами и агентами реального времени: мгновенный агент захватывает свой ресурс сразу, как только $NumC = 0$, в то время как агент реального времени, которому требуется время, чтобы арендовать ресурс, хочет убедиться, что за это время никакой другой агент не планирует пользоваться этим же ресурсом. $NumC$ представляет собой верхнюю оценку агентом количества партнёров-агентов, с которыми у него может быть конфликт, и, соответственно, $CFree$ — это его текущий счётчик количества партнёров, которые в течение определённого времени подтверждали, что по их мнению, у них нет конфликтов ни с кем. Формально:

- Агент реального времени считает (имеет мнение), что он не конфликтует ни с одним другим агентом, если его счётчик $NumC = 0$,
- Агент реального времени считает (имеет мнение), что в системе нет конфликтов, как только $NumC = 0$ и $CFree = 2(n - 1)$, т.е. он дважды проверяет, что все остальные агенты полагают, что у них тоже нет конфликтов.

Если два агента конкурируют, то они решают конфликт с помощью функции $reSOL : Resources \times \mathbb{N} \times \mathbb{N} \times ResourceList \longrightarrow Resources \cup \{null\}$. Конфликтующие агенты вычисляют значение этой функции, используя данные о времени прибытия друг друга в центр ресурсов и приоритеты своих желаний: в конкуренции за ресурс выигрывает тот агент, который раньше прибыл в центр, и его намерение остаётся прежним, а проигравший агент выбирает в качестве намерения следующее по приоритету желание из своего списка. Отметим, что у проигравшего агента нет шансов когда-либо выиграть упущенный ресурс, т.к. если выигравший агент и уступит его впоследствии, то только тому, кто прибыл ещё раньше него. Поэтому проигранный ресурс удаляется из списка желаний проигравшего. В случае, если список желаний исчерпан, то значением функции $reSOL$ является $null$ и агент уходит из центра ресурсов. Формально, для всех агентов i и j ($i \neq j$), для времени прибытия t_i и t_j ($t_i \neq t_j$), для любых ресурсов u_i и u_j, и для списка желаний агента U_i:

- если $CONF(i, u_i, j, u_j)$, то
  - если $t_i < t_j$, то $reSOL(u_i, t_i, t_j, U_i) = u_i$;
  - если $t_i > t_j$, то $reSOL(u_i, t_i, t_j, U_i) = car(U_i)$;
- если $reSOL(u_i, t_i, t_j, U_i) = null$, то агент i выбывает из множества $POOL$.

Здесь и далее функция $car(List)$ возвращает первый элемент списка $List$ и удаляет его из этого списка.

В псевдокоде алгоритма мы используем следующие приватные переменные:

- `Me` — целочисленный идентификатор агента;
- `Desires` — целочисленный упорядоченный список желаний агента;
- `cur_des` — переменная для текущего списка желаний агента;
- `cur_un` и `par_un` — целочисленные переменные текущего намерения агента и намерения партнёра, с которым агент в настоящее время ведёт переговоры;
- `time` и `par_time` — целочисленное время регистрации агента и его партнёра в центре ресурсов;
- `par_bel` — булева переменная для мнения текущего партнёра о том, что у него нет конфликтов;

- contacts — переменная для (неупорядоченного) множества агентов, с которыми должны проводиться переговоры.

Описание алгоритма и псевдокод

Функционирование агента состоит из серии раундов связи. В каждом раунде агент должен пообщаться со всеми другими агентами. В каждом стандартном раунде, когда агент связывается с другим агентом, он проверяет, конфликтуют ли они за элемент ресурса. Всякий раз, когда агент не конфликтует с текущим партнёром, он уменьшает значение своей приватной переменной $NumC$, в противном случае он повторно инициализирует значение $NumC$ числом $(n-1)$, и кроме того, если агент проигрывает конкуренцию за ресурс, то его намерением становится очередное желание, а сам список его желаний убывает. Если же список желаний исчерпан, то агент завершает свою работу, покидая центр ресурсов и уменьшая при этом множество $POOL$.

Если стандартный раунд заканчивается с $NumC = 0$, т. е. агент полагает, что у него нет конфликтов, то он начинает раунды двойной проверки. В этих раундах он должен убедиться, что другие агенты тоже считают, что они бесконфликтны. Всякий раз при двойной проверке, когда партнёр уведомляет, что он полагает себя свободным от конфликтов, агент увеличивает значение своей приватной переменной $CFree$. Но если партнёр сообщает, что он не может считать, что у него нет конфликтов, то агент возвращается к стандартным раундам, повторно инициализируя переменную $NumC$ числом $(n-1)$ и переменную $CFree$ числом 0. Эти раунды называются двойной проверкой, потому что агент останавливается и берёт, наконец, в аренду последний выбранный ресурс, когда $CFree = 2(n-1)$, т.е. когда каждый из остальных агентов дважды подтвердит своё мнение о том, что у него нет конфликтов.

Мы приводим алгоритм для агентов реального времени. Алгоритм для мгновенных агентов получается игнорированием счётчика $CFree$.

```
const Me : integer in [1..n];
const time : integer;
const Desires : list of [1..m];
var NumC : integer in [1..(n-1)];
var CFree : integer in [1..(n-1)];
var contacts : set of [1..n];
var partner : integer in [1..n];
var cur_un, par_un : integer in [1..m];
var cur_des : list of [1..m];
var par_time : integer;
var par_bel : boolean;
begin
1: NumC := (n - 1); CFree := 0;
2: cur_des := Desires; cur_un := car(cur_des);
3: repeat
4:    if NumC > 0 then NumC := (n-1);
5:    contacts := POOL\{Me};
6:    repeat
7:      partner := any in the contacts ready to communicate;
8:      start communication with the partner:
9:      { send (<cur_un>; time; <(NC=0)?>) to partner
10:       receive (<par_un>; par_time; <par_bel>) from partner }
11:       if CONF(Me, cur_un, partner, par_un)
12:       then { cur_un := reSOL(cur_un, time, par_time, cur_des);
```

```
13:                if cur_un = null then goto leave;
14:                NumC := (n-1); CFree:= 0 }
15:       else if NumC > 0
16:            then { NumC := (NumC-1); CFree := 0}
17:            else if par_bel
18:                 then CF := CF+1
19:                 else { NumC := (n-1); CFree := 0 }
20:      close communication session with partner; }
21:      contacts := contacts\partner;
22:    until ( contacts != empty )
23: until ( NumC = 0 & CFree = 2(n-1) )
24: leave: POOL := POOL\{Me};
end.
```

Корректность и анализ решения

Утверждение 2. *Предположим, что в системе гарантирована справедливость коммуникаций между агентами и завершение работы. Тогда в момент завершения работы каждый зарегистрированный агент будет знать, что на ресурс, на который он претендует, не претендует никакой другой агент.*

Вполне упорядоченное множество — это частичный порядок $(D, \leq)$ без бесконечных строго убывающих последовательностей. Мы говорим, что распределённая система является вполне упорядоченной, если существует вполне упорядоченное множество $(D, \leq)$ и отображение F из конфигураций системы в D такое, что для всех агентов $i \neq j$, для всех элементов ресурса u_i и u_j, $CONF(i, u_i, j, u_i)$ означает, что выполнение $reSOL(u_i, t_i, t_j, U_i)$ уменьшает значение F.

Утверждение 3. *Предположим, что в системе гарантирована справедливость коммуникаций между агентами и система является вполне упорядоченной. Тогда система рано или поздно обязательно завершает свою работу.*

Наша мультиагентная система, реализующая вышеприведённый алгоритм, является вполне упорядоченной. В качестве функции F можно взять общее количество текущих желаний агентов. Действительно, в случае конкуренции за ресурс, проигравший агент уменьшает количество своих желаний вплоть до *null*.

Первая проблема с основанным на знаниях мультиагентным протоколом гонки за ресурсы — это отказоустойчивость: если какой-либо из агентов откажется от предписанного поведения, результат будет неверным. Вторая и основная проблема — это сложность этого алгоритма: наилучшая известная нам верхняя граница количества раундов равна $n!$ и в каждом раунде содержится $O(n^2)$ сеансов взаимодействия между агентами.

Заключение

Фактически, рассмотренные в настоящей статье мультиагентные алгоритмы, основанные на знаниях — это алгоритмы достижения своего рода консенсуса и, возможно, общего знания [7] при сохранении приватности: мнение (убеждение) каждого агента становится знанием (имеет доказательство) и каждый агент знает, что все другие агенты знают, что если какой-либо агент совершает действие, то этот совершающий действие агент имеет на это право. Для нас представляет интерес задача о достижимости не только консенсуса, но общего знания об этом консенсусе, но эта задача требует дополнительного исследования.

Надо сказать, что нам известно очень мало литературы по мультиагентным алгоритмам, основанных на знаниях. Фактически, мы можем указать только на фундаментальную монографию

[7]. Правда, есть довольно-таки обширная литература по различным вариантам динамической эпистемической логики для мультиагентных систем (например, [11, 12]), но эта литература посвящена классическим логическим вопросам — семантика и аксиоматизируемость, выразительная сила и разрешимость — но не вопросам конструирования мультиагентных алгоритмов, основанных на знаниях.

Из нашей статьи можно сделать вывод, что мультиагентные алгоритмы зачастую неэффективны и немасштабируемы: согласованность (достижение консенсуса, например), доступность (системы для новых агентов, желающих принять участие в ее работе) и высокая производительность (например, время, необходимое для достижения консенсуса) не достижимы одновременно. Например, в алгоритме гонки за ресурсы именно получение итоговых знаний с соблюдением приватности приводит к снижению производительности системы. Но мы хотели бы закончить статью на более оптимистической ноте: необходимы дополнительные исследования (включая моделирование) мультиагентного алгоритма гонки за ресурсы, и исследования достижения частичного консенсуса (с определённой вероятностью).

References

[1] M. Takada, *Distributed Systems: for Fun and Profit*. 2013. [Online]. Available: http://book.mixu.net/distsys/.

[2] A. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2006.

[3] M. Wooldridge, *An Introduction to Multiagent Systems*. John Willey&Sons, 2002.

[4] C. Chappell, *Stanford Encyclopedia of Philosophy*. 2019, ch. Plato on Knowledge in the Theaetetus. [Online]. Available: http://plato.stanford.edu/entries/plato-theaetetus/.

[5] J. Ichikawa and M. Steup, *Stanford Encyclopedia of Philosophy*. 2017, ch. The Analysis of Knowledge. [Online]. Available: http://plato.stanford.edu/entries/knowledge-analysis/.

[6] P. Dütting and A. Geiger, *Algorithmic Mechanism Design*. Seminar Report, University of Karlsruhe, Fakultät für Informatik, 2007. [Online]. Available: https://webspace.science.uu.nl/~leeuw112/msagi/mech_design.pdf.

[7] R. Fagin, J. Halpern, Y. Moses, and M. Vardi, *Reasoning about Knowledge*. MIT Press, 1995.

[8] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145, 2011. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[9] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 2012.

[10] A. Satekbayeva and N. Shilov, “Some Results on Multiagent Algorithms in Social Computing/Software Context”, *Information*, vol. 17, no. 1, pp. 229–240, 2014.

[11] J. van Benthem, *Logical Dynamics of Information and Interaction*. Cambridge University Press, 2011.

[12] N. Alechina and B. Logan, “State of the Art in Logics for Verification of Resource-Bounded Multi-Agent Systems”, in *Fields of Logic and Computation III — Essays Dedicated to Yuri Gurevich on the Occasion of His 80th Birthday*, ser. LNCS, vol. 12180, Springer, 2020, pp. 9–29.