Верификация алгоритма кольца для распределённых систем с использованием языка спецификаций временной логики действий

В.И. Шиян, А.М. Кузнецов, П.Н. Литвиненко, А.А. Полтавская, М.А. Прохоров, К.А. Степуленко, Д.С. Токарева

Кубанский государственный университет, Краснодар

Аннотация: Статья посвящена проблеме верификации распределённых алгоритмов с использованием формальных методов. В качестве объекта исследования выбран классический алгоритм выбора лидера в кольцевой топологии – алгоритм кольца. Для его анализа применяется язык спецификаций временной логики действий (Temporal Logic of Actions – TLA+). В работе представлена детальная формальная модель алгоритма, описывающая его состояния и переходы с учётом особенностей распределённых систем, таких как отсутствие разделяемой памяти. Формулируются и доказываются ключевые свойства корректности: уникальность лидера (свойство безопасности), завершаемость выборов (свойство живости) и согласие. Корректность спецификации подтверждена с помощью модельного верификатора моделей языка временной логики действий, который исчерпывающе проверил все достижимые состояния для модели с тремя процессами. Результаты демонстрируют эффективность языка спецификаций временной логики действий (TLA+) для обеспечения высокой степени уверенности в надёжности распределённых систем.

Ключевые слова: формальная верификация, распределённые системы, алгоритм кольца, выбор лидера, язык спецификаций временной логики действий, проверка моделей, свойства безопасности, свойства живости.

Введение

Проблема верификации распределённых систем является одной из наиболее сложных и актуальных задач современной информатики. Распределённая система представляет собой совокупность автономных вычислительных процессов, взаимодействующих путём обмена сообщениями [1]. Сложность таких систем обусловлена недетерминизмом, асинхронностью коммуникаций и возможностью отказов, что порождает огромное пространство состояний и делает невозможным исчерпывающее тестирование [2]. Одной из фундаментальных задач в таких системах является выбор лидера – достижение консенсуса между процессами относительно того, какой из них будет выполнять координирующую роль. Алгоритм кольца является классическим решением этой задачи для систем с

кольцевой топологией. Однако его кажущаяся простота скрывает тонкости, связанные с конкуренцией и асинхронностью. Для обеспечения гарантий корректной работы необходимо применение формальных методов, позволяющих математически строго доказать соответствие алгоритма требуемым свойствам. В данной работе проводится исчерпывающий анализ и верификация алгоритма кольца с помощью языка спецификаций временной логики действий (Temporal Logic of Actions – TLA+) [3].

Постановка задачи и описание алгоритма

Формальная спецификация требует определения математической модели системы и задачи выбора лидера. Распределённая система моделируется как граф, где узлы – это процессы, а рёбра – каналы связи. В контексте алгоритма кольца рассматривается топология, где процессы образуют однонаправленное кольцо. Для каждого процесса в кольце однозначно определён его «сосед», которому он может отправлять сообщения. Каждый процесс обладает уникальным идентификатором [4], принадлежащим вполне упорядоченному множеству положительных целых чисел. Задача выбора лидера требует, чтобы алгоритм удовлетворял трём фундаментальным свойствам: уникальности (свойство безопасности) – в любой момент времени существует не более одного лидера, завершаемости (свойство живости) – в конечном итоге лидер будет избран, и согласия (свойство безопасности и живости) – все процессы в конечном итоге узнают одного и того же лидера.

Алгоритм кольца, предложенный Ле Ланом [5], функционирует следующим образом. Любой процесс, которому требуется лидер, инициирует выборы, создавая сообщение со своим уникальным идентификатором и отправляя его соседу. При получении сообщения процесс сравнивает содержащийся в нём идентификатор со своим. Если полученный идентификатор больше, процесс пересылает сообщение дальше. Если

меньше, сообщение отбрасывается. Ключевой момент наступает, когда процесс получает сообщение, содержащее его собственный идентификатор. Это означает, что его сообщение обошло всё кольцо, не встретив процесса с большим идентификатором. В этот момент данный процесс объявляет себя лидером и инициирует рассылку второго типа сообщения, уведомляющего всех участников о его избрании. Получив такое уведомление, каждый процесс записывает идентификатор лидера и пересылает уведомление дальше [6, 7].

Формальная модель в TLA+

Для формализации и верификации данного алгоритма используется TLA+. TLA+ – это математический язык [8] для спецификации и анализа конкурентных и распределённых систем. Система в TLA+ представляется как множество всех возможных её поведений, где поведение – это бесконечная последовательность состояний. Полная спецификация поведения системы определяется как логическая формула, утверждающая, что система должна начинаться в одном из начальных состояний, определённых предикатом Init, и впоследствии каждый её шаг должен соответствовать правилам перехода, описанным в формуле Next. Это требование должно выполняться всегда, что в темпоральной логике обозначается специальным оператором.

Формальная модель алгоритма определяется набором переменных состояния. Константа N задаёт число процессов, а функция уникального идентификатора (Unique Identifier – UID) отображает индекс процесса в его уникальный идентификатор. Состояние системы описывается переменными: messages – множество сообщений в сети, state[i] – текущий статус процесса i (например, «пассивный», «кандидат», «лидер»), leader_known[i] – локальное знание процесса i о лидере, max_seen_id[i] – максимальный идентификатор, увиденный процессом i. Использование leader_known[i] вместо единой

глобальной переменной точно моделирует распределённую природу системы без разделяемой памяти.

Начальное состояние и правила перехода

Начальное состояние системы, описываемое предикатом Init, определяется следующим образом. Все процессы находятся в пассивном состоянии, ни один из них не участвует в выборах лидера. Сеть пуста, то есть отсутствуют какие-либо сообщения между процессами. Переменные, хранящие максимальный увиденный идентификатор, для каждого процесса инициализируются нулём. Для обозначения отсутствия знания о лидере используется числовое значение ноль, поскольку идентификаторы процессов являются строго положительными числами. Такой подход обеспечивает типобезопасность модели, избегая смешения числовых и строковых типов данных, и является важным уточнением, продиктованным строгостью формального метода. Благодаря этому достигается однозначность в трактовке значения переменных и предотвращается появление неоднозначных ситуаций при анализе состояний системы.

Динамика системы описывается формулой перехода Next, которая является объединением (дизъюнкцией) всех возможных действий для всех процессов. Каждое действие представляет собой атомарный шаг и включает три возможных варианта. Первый вариант – инициирование выборов: процесс в пассивном состоянии может перейти в состояние кандидата и отправить сообщение о начале выборов своим соседям по сети. Второй вариант – обработка сообщения: процесс получает сообщение. Если это сообщение о выборах, он сравнивает идентификаторы. При превосходстве чужого идентификатора сообщение пересылается дальше, при превосходстве собственного идентификатора оно отбрасывается. Если процесс получает обратно свой собственный идентификатор, он становится лидером, обновляет своё локальное знание о лидере и рассылает уведомление об этом

остальным процессам. Если процесс получает уведомление об уже избранном лидере и сам о нём ещё не знает, он обновляет своё локальное знание и пересылает уведомление дальше по сети. Третий вариант – «заикание», когда процесс ничего не делает, и глобальное состояние системы не меняется.

На рис. 1 показаны объявление модуля, определения констант и переменных, используемых в спецификации.

```
------------------------- MODULE RingAlgorithm -------------------------
EXTENDS Naturals, Sequences, TLC

N == 3
UID == [p \in 1..N |-> p]
Procs == 1..N

VARIABLES messages,
          state,
          leader_known,
          max_seen_id,
          pc

vars == << messages, state, leader_known, max_seen_id, pc >>

Succ(p) == IF p = N THEN 1 ELSE p + 1
```

Рис. 1. – Объявление модуля, константы и переменные

На рис. 2 представлена формула инициализации Init, задающая начальное состояние системы.

```
Init == /\ messages = {}
        /\ state = [p \in Procs |-> "passive"]
        /\ leader_known = [p \in Procs |-> 0]
        /\ max_seen_id = [p \in Procs |-> 0]
        /\ pc = [self \in Procs |-> "P_Loop"]
```

Рис. 2. – Формула инициализации

На рис. 3 показана первая часть определения действия процесса, описывающая переход из пассивного состояния в кандидаты.

```
p(self) == /\ pc[self] = "P_Loop"
           /\ (
                 ( /\ state[self] = "passive"
                   /\ state' = [state EXCEPT ![self] = "candidate"]
                   /\ max_seen_id' = [max_seen_id EXCEPT ![self] = UID[self]]
                   /\ messages' = messages \union {[type |-> "election", val |->
UID[self], dest |-> Succ(self)]}
                   /\ UNCHANGED <<leader_known, pc>> )
```

Рис. 3. – Первая часть действия процесса, переход из пассивного состояния

На рис. 4 представлена вторая часть действия процесса, описывающая обработку входящих сообщений различных типов.

```
\/
    ( /\ \E m \in {msg \in messages : msg.dest = self}:
        /\ IF m.type = "election"
            THEN
                IF m.val > UID[self]
                THEN
                        /\ state' = [state EXCEPT ![self] = "candidate"]
                        /\ max_seen_id' = [max_seen_id EXCEPT ![self] = m.val]
                        /\ messages' = (messages \ {m}) \union {[type |->
"election", val |-> m.val, dest |-> Succ(self)]}
                        /\ UNCHANGED <<leader_known, pc>>
                ELSE IF m.val = UID[self]
                        THEN
                            /\ state' = [state EXCEPT ![self] = "leader"]
                            /\ leader_known' = [leader_known EXCEPT ![self] =
UID[self]]
                            /\ messages' = (messages \ {m}) \union {[type |->
"elected", val |-> UID[self], dest |-> Succ(self)]}
                            /\ UNCHANGED <<max_seen_id, pc>>
                        ELSE
                            /\ messages' = messages \ {m}
                            /\ UNCHANGED <<state, leader_known, max_seen_id, pc>>
            ELSE
                IF leader_known[self] = 0
                THEN
                        /\ state' = [state EXCEPT ![self] = "non_leader"]
                        /\ leader_known' = [leader_known EXCEPT ![self] = m.val]
                        /\ messages' = (messages \ {m}) \union {[type |->
"elected", val |-> m.val, dest |-> Succ(self)]}
                        /\ UNCHANGED <<max_seen_id, pc>>
                ELSE
                        /\ messages' = messages \ {m}
                        /\ UNCHANGED <<state, leader_known, max_seen_id, pc>> )
```

Рис. 4. – Вторая часть действия процесса, обработка сообщений

На рис. 5 показана завершающая часть действия процесса и определения основных формул спецификации.

```
\/
                ( /\ UNCHANGED vars )
             )

Next == \E self \in Procs: p(self)

Spec == Init /\ [][Next]_vars

Fairness == WF_vars(Next)

FairSpec == Spec /\ Fairness

Uniqueness == \A p1, p2 \in Procs: (state[p1] = "leader" /\ state[p2] = "leader")
=> (p1 = p2)
Termination == <>(\E proc \in Procs : state[proc] = "leader")
AgreementLiveness == [](((\E proc \in Procs : state[proc] = "leader") => <>(\A q
\in Procs: leader_known[q] # 0))
```

Рис. 5. – Третья часть действия процесса и остальные формулы

Верификация свойств корректности

После получения формальной спецификации производится её верификация с помощью модельного верификатора языка спецификаций временной логики действий (TLA+ Model Checker – TLC). Проверяются ключевые свойства корректности.

Свойство безопасности: уникальность лидера

Проверяется свойство безопасности – уникальность лидера. Оно формулируется как инвариант состояния, то есть утверждение, которое должно быть истинным в каждый момент времени. Этот инвариант гласит, что для любых двух процессов в системе, если оба они находятся в состоянии «лидер», то эти два процесса должны быть одним и тем же.

Математическое доказательство уникальности основано на полном порядке идентификаторов. Допустим, существуют два различных лидера. Это означает, что два разных сообщения о выборах (с меньшим и большим

идентификатором) должны были успешно обойти всё кольцо. Однако сообщение с меньшим идентификатором на своём пути неизбежно встретило бы процесс с большим идентификатором. Согласно правилам алгоритма, такой процесс отбросил бы это сообщение, не переслав его дальше. Следовательно, сообщение с меньшим идентификатором никогда бы не завершило обход, что приводит к противоречию. Таким образом, лидером может стать только процесс с максимальным идентификатором среди всех, инициировавших выборы.

Свойство живости: завершаемость выборов

Далее верифицируется свойство живости – завершаемость выборов. Оно утверждает, что в любой возможной последовательности событий рано или поздно наступит момент, когда один из процессов перейдёт в состояние «лидер». Для корректной проверки таких свойств, утверждающих, что «что-то хорошее в итоге случится», необходимо ввести предположение о справедливости [9]. Это предположение гарантирует, что ни один процесс, способный совершить действие, не будет игнорироваться вечно. В модели это формализуется специальным условием, которое отсекает все нереалистичные, «нечестные» сценарии выполнения, где система бездействует без причины.

Свойство согласия

Наконец, проверяется свойство согласия. Оно состоит из двух частей. Аспект безопасности утверждает, что если два любых процесса знают, кто является лидером (то есть их локальное знание отлично от нуля), то они должны быть согласны в его идентификаторе. Аспект живости утверждает, что если лидер в системе был избран, то в конечном итоге каждый процесс в системе узнает о том, кто является лидером. Оба эти свойства также доказываются математически, исходя из того, что уведомительное

сообщение создаётся единственным лидером и при условии справедливости гарантированно достигает всех узлов в кольце.

Заключение

В заключение, использование TLA+ позволило создать строгую и точную спецификацию алгоритма выбора лидера в кольце. TLC был применён для проверки этой спецификации на конкретной модели с тремя процессами. Проверка, в ходе которой были исчерпывающе исследованы все 44 достижимых состояния, успешно завершилась, подтвердив, что спецификация удовлетворяет всем заявленным свойствам безопасности и живости, включая уникальность, завершаемость и согласие. Этот процесс демонстрирует, как формальные методы обеспечивают высокий уровень уверенности в корректности распределённых алгоритмов, превосходящий возможности традиционного тестирования [10]. Такой подход служит основой для построения надёжных и предсказуемых распределённых систем [11, 12].

Литература

1. Таненбаум Э., ван Стеен М. Распределённые системы. Принципы и парадигмы. СПб.: Питер. 2003. 876 с.

2. Карпов Ю.Г. Model checking. Верификация параллельных и распределённых программных систем. СПб.: БХВ-Петербург. 2010. 560 с.

3. Lamport L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Reading, MA: Addison-Wesley. 2002. 384 p.

4. Garcia-Molina H. Elections in a distributed computing system. IEEE Transactions on Computers. 1982. V. C-31. No. 1. Pp. 48–59.

5. Le Lann G. Distributed systems—towards a formal approach. Information Processing 77. IFIP Congress. Amsterdam: North-Holland. 1977. Pp. 155–160.

6. Chang E., Roberts R. An improved algorithm for decentralized extrema-finding in circular configurations of processes. Communications of the ACM. 1979. V. 22. No. 5. Pp. 281–283.

7. Hirschberg D.S., Sinclair J.B. Decentralized extrema-finding in circular configurations of processors. Communications of the ACM. 1980. V. 23. No. 11. Pp. 627–628.

8. Lamport L. The Temporal Logic of Actions. ACM Transactions on Programming Languages and Systems. 1994. V. 16. No. 3. Pp. 872–923.

9. Baier C., Katoen J.-P. Principles of Model Checking. Cambridge, MA: MIT Press. 2008. 975 p.

10. Clarke E.M., Grumberg O., Peled D. Model Checking. Cambridge, MA: MIT Press. 1999. 314 p.

11. Бурякова Н.А., Чернов А.В. Классификация частично формализованных и формальных моделей и методов верификации программного обеспечения // Инженерный вестник Дона. 2010. №4. URL: ivdon.ru/magazine/archive/n4y2010/259/.

12. Кривошеев М.О., Логвинов Ю.Н. Протокол консенсуса в асинхронных сетях с неисправностями // Инженерный вестник Дона. 2013. №4. URL: ivdon.ru/magazine/archive/n4y2013/2154/.

References

1. Tanenbaum E., van Steen M. Raspredelennye sistemy. Printsipy i paradigmy [Distributed Systems. Principles and Paradigms]. SPb.: Piter, 2003. 876 p.

2. Karpov Yu.G. Model checking. Verifikatsiya parallel'nykh i raspredelennykh programmnykh sistem [Model Checking. Verification of Parallel and Distributed Software Systems]. SPb.: BHV-Peterburg, 2010. 560 p.

3. Lamport L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Reading, MA: Addison-Wesley, 2002. 384 p.

4. Garcia-Molina H. IEEE Transactions on Computers. 1982, C-31(1), Pp. 48–59.

5. Le Lann G. Information Processing 77. IFIP Congress. Amsterdam: North-Holland, 1977. Pp. 155–160.

6. Chang E., Roberts R. Communications of the ACM. 1979, 22(5), Pp. 281–283.

7. Hirschberg D.S., Sinclair J.B. Communications of the ACM. 1980, 23(11), Pp. 627–628.

8. Lamport L. ACM Transactions on Programming Languages and Systems. 1994, 16(3), Pp. 872–923.

9. Baier C., Katoen J.-P. Principles of Model Checking. Cambridge, MA: MIT Press, 2008. 975 p.

10. Clarke E.M., Grumberg O., Peled D. Model Checking. Cambridge, MA: MIT Press, 1999. 314 p.

11. Buryakova N.A., Chernov A.V. Inzhenernyj vestnik Dona. 2010. №4. URL: ivdon.ru/magazine/archive/n4y2010/259/.

12. Krivosheev M.O., Logvinov Yu.N. Inzhenernyj vestnik Dona. 2013. №4. URL: ivdon.ru/magazine/archive/n4y2013/2154/.

Дата поступления: 12.11.2025

Дата публикации: 25.12.2025