УДК 004.75

А. В. Ильин [1], **Б. Н. Пищик** [2]

[1] *Новосибирский государственный университет*
ул. Пирогова, 2, Новосибирск, 630090, Россия

[2] *Конструкторско-технологический институт вычислительной техники СО РАН*
ул. Акад. Ржанова, 6, Новосибирск, 630090, Россия

sanekspot@gmail.com, boris.pishchik@kti.nsc.ru

МЕТОДЫ РЕПЛИКАЦИИ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ

Рассмотрены различные методы репликации объектов в ненадежной распределенной вычислительной среде. Два основных подхода – это использование state-machine и репликация с первичной и вторичной копиями. Описано несколько алгоритмов реализации state-machine метода. Проведено сравнение методов по единым критериям.

Ключевые слова: распределенная система, репликация, отказоустойчивость, state-machine, алгоритм консенсуса.

Введение

В настоящее время требования к программным системам, а также ответственность, возложенная на них, все возрастают. Ни одна из сфер деятельности человека, таких как медицина, финансы, промышленность, образование, наука, уже немыслима без применения сложных информационных систем. Выход системы из строя или потеря критических данных могут привести к нарушению бизнес-процессов и нанести существенный ущерб для общества. В связи с этим к надежности современных систем предъявляются повышенные требования: они должны иметь высокий коэффициент доступности («коэффициент готовности» согласно стандарту ГОСТ [1]) и обладать свойством отказоустойчивости (в стандарте ГОСТ [1] используется менее распространенный термин «устойчивость к неисправности»).

Насколько бы надежным ни было программное средство, оно всегда зависит от среды, в которой функционирует. В связи с тем что обеспечить стопроцентную надежность технических средств не представляется возможным, отказоустойчивость систем обеспечивается программными средствами резервирования. Такие средства опираются на отказоустойчивые алгоритмы, а также на наличие и доступность данных, необходимых для восстановления корректной работы системы после отказа части оборудования.

Одним из естественных способов осуществить резервирование информационного объекта является поддержка нескольких его копий в системе. Механизм, реализующий копирование и синхронизацию содержимого копий, называется репликацией.

В статье приводится обзор существующих подходов организации механизма репликации в распределенных системах, выделяются их основные преимущества и недостатки.

Ильин А. В., Пищик Б. Н. Методы репликации в распределенных системах // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2016. Т. 14, № 2. С. 52–58.

ISSN 1818-7900. Вестник НГУ. Серия: Информационные технологии. 2016. Том 14, № 2

Модель

Распределенная система представляется как совокупность автономных компьютеров или процессоров и коммуникационной подсистемы. Компьютеры или процессоры будем называть узлами распределенной системы, на которых исполняются информационно-вычислительные процессы, оперирующие информационными объектами. Обмен данными между узлами происходит путем отправки сообщений через коммуникационную подсистему. Передача сообщений происходит асинхронно [2].

Информационный объект, который необходимо резервировать, – это набор байтов, представляющий состояние этого объекта, и связанный с ним набор операций, позволяющий модифицировать это состояние. В общем случае, операции могут быть детерминированными (результат зависит только от текущего состояния объекта и параметров операции) либо недетерминированными. Отметим, что под данное определение объекта попадают как просто данные, над которыми можно производить операции чтения и записи, так и целые программы, с которыми можно взаимодействовать посредством удаленных вызовов.

Каждому информационному объекту соответствует единственный процесс, который может проводить операции над ним. Назовем такой процесс сервером объекта. Все процессы, желающие оперировать объектом, должны направлять запросы серверу посредством передачи сообщения. Процессы, взаимодействующие с объектом посредством запросов, называются клиентами этого объекта.

Если объект является критически важным для работы всей распределенной системы, то в системе необходимо поддерживать несколько его копий, причем эти копии должны быть *эквивалентны*. Копии информационного объекта также будем называть репликами. Две реплики одного объекта считаются эквивалентными, если наборы байтов, описывающие их состояния, полностью совпадают.

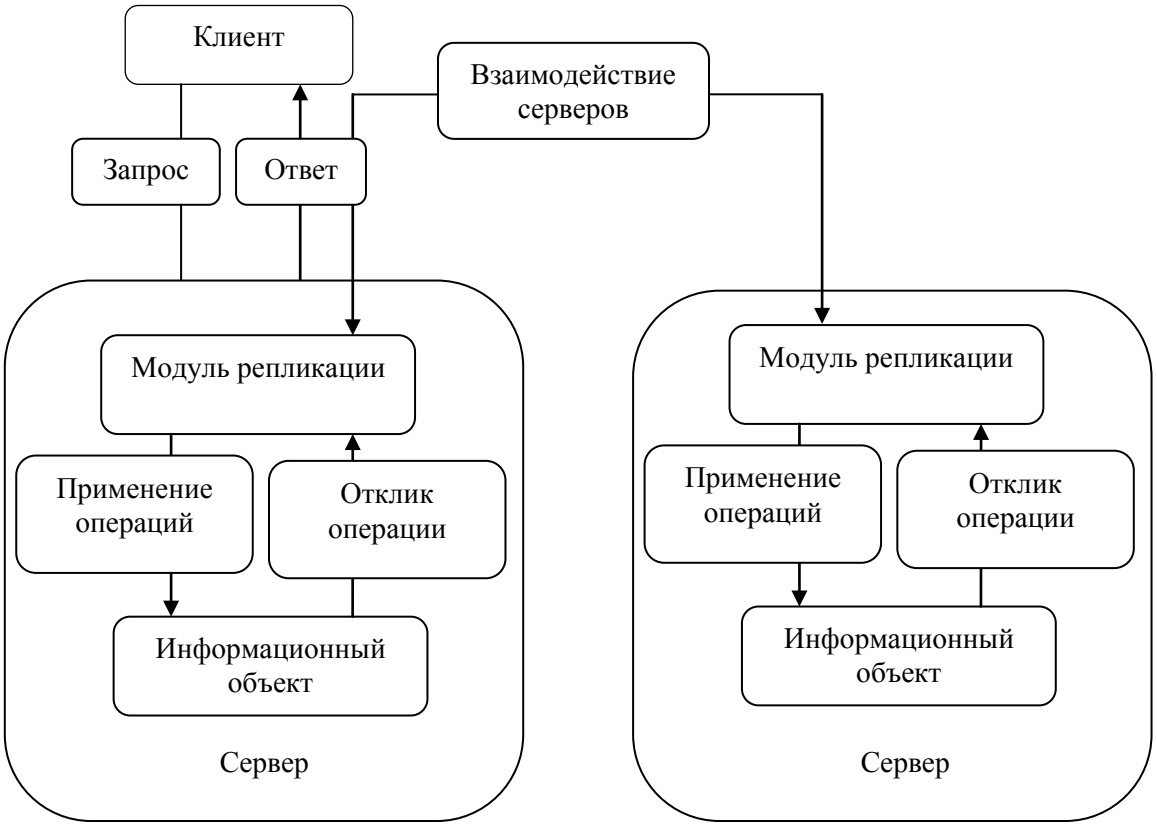
Посредством применения операций состояние объекта со временем может измениться, и это может произойти независимо на каждом сервере при отсутствии какой-либо синхронизации. Целью репликации является синхронизация состояний копий объекта во всех узлах распределенной системы. Во время работы алгоритма репликации состояние копий объекта на разных серверах может быть различным, важно, чтобы от пользователя объекта эти различия были скрыты. Данный принцип носит название «прозрачность репликации» [3]. Он говорит о том, что «для пользователей должна быть создана такая среда, чтобы они, по крайней мере, с логической точки зрения могли считать, что данные не дублируются».

Каждый сервер содержит модуль репликации, выполняющий все задачи, связанные с синхронизацией копий. Запросы от клиентов также обрабатываются модулем репликации. Схема взаимодействия клиентских и серверных процессов приведена на схеме (см. рисунок).

Задача репликации осложняется тем, что узлы могут быть подвержены неисправностям. В качестве модели неисправностей примем модель выхода процесса из строя [2]. Согласно данной модели, процесс до некоторого момента работает правильно (согласно его спецификации), а затем перестает совершать какие-либо действия и отвечать на сообщения. Остальные процессы могут обнаружить неисправность с помощью таймаутов.

Рассмотрим следующие подходы к решению проблемы репликации. Будем считать, что имеется только один информационный объект, который необходимо реплицировать. Один из подходов основывается на том, что все копии объекта модифицируются одновременно (state-machine replication) [4]. Другой подход состоит в том, чтобы объявить одну из копий как первичную, и производить модификации в первую очередь над ней, а все остальные копии обновлять асинхронно [3].

Введем обозначения. Пусть x – информационный объект, P – множество процессов, $S \subseteq P$ – множество серверов, на которых располагаются копии объекта x, $C \subseteq P$ – множество клиентов объекта x. Конкретные представители множеств будут обозначаться соответствующими строчными латинскими буквами. Соответствующие множества исправных процессов обозначаются заглавной буквой с индексом up, например P_{up} – множество исправных процессов.



Взаимодействие процессов

State machine replication

Главной проблемой при реализации state-machine репликации является необходимость упорядочивания клиентских запросов между всеми серверами. Наличие единого порядка является обязательным условием для этого метода [4]. Строго говоря, нам необходимо задать некоторый строгий линейный порядок на множестве клиентских запросов $\mathcal{M}$, в котором эти запросы будут обрабатываться серверами, так чтобы $\forall m_1, m_2 \in \mathcal{M} : m_1 \neq m_2$ выполняется $m_1 < m_2$ или $m_2 < m_1$. Пусть $H_s \subseteq \mathcal{M}$ – множество запросов, полученных сервером s, $H_{s,c} \subseteq H_s$ – множество запросов, полученных сервером s от клиента c. Будем говорить, что запрос $m \in \mathcal{M}$ является *стабильным* для сервера s, если $\forall m' \in (\mathcal{M} \setminus H_s)$ верно неравенство $m < m'$. Если все серверы будут обрабатывать только стабильные запросы в соответствии с заданным порядком, и каждый сервер получит каждый запрос, то порядок обработки запросов будет одинаковым для всех серверов [4].

Рассмотрим различные способы задания порядка на множестве клиентских запросов, основная часть которых изложена в [4]:

- Логические часы.
- Синхронизованные часы реального времени.
- Идентификаторы, генерируемые серверами.
- Алгоритм консенсуса.

Логические часы – это отображение $\hat{T}: e \to \mathbb{N}$, которое сопоставляет каждому событию в системе некоторое натуральное число [5]. Для двух различных событий e и e' должно вы-

полняться условие $\hat{T}(e) < \hat{T}(e')$ или $\hat{T}(e') < \hat{T}(e)$. Если появление события e' зависит от события e, то верно неравенство $\hat{T}(e) < \hat{T}(e')$.

Для реализации логических часов в распределенных системах с каждым процессом p связывается счетчик $\hat{T}_p$, значение которого $\hat{T}_p(m)$ передается вместе со всеми сообщениями m, отправляемыми p. Предполагается, что в любой момент времени каждому серверу известно множество исправных клиентов $C_{up} \subseteq C$.

При использовании логических часов запрос m считается стабильным для сервера s, если $\forall c \in C_{up}\ \exists m' \in H_{s,c} : \hat{T}_c(m) < \hat{T}_c(m')$.

Следует отметить, что данный способ возлагает ответственность за репликацию на клиентов. Клиенты должны отправлять свои запросы всем серверам, кроме того, каждый клиент периодически должен посылать heartbeat-сообщения всем серверам для того, чтобы осуществлялся прогресс системы.

Добиться упорядочивания клиентских запросов также можно с помощью *синхронизированных часов реального времени* [6]. За $T_p(t)$ обозначим значение на часах процесса p в момент времени t, $t(e)$ будет обозначать реальное время возникновения события e. Предположим, что все часы в системе синхронизированы с точностью δ. Это значит, что $\forall p_1, p_2 \in P\ \forall t\ |T_{p_1}(t) - T_{p_2}(t)| < \delta$. Для данного алгоритма требуется, чтобы δ было меньше, чем минимальное время доставки сообщения от одного узла другому. Клиент c рассылает каждый запрос m всем серверам вместе с временной отметкой $T_c(t(m))$.

Также необходимо существование константы Δ такой, что для запроса m, отправленного клиентом c, выполняется $\forall s \in S_{up}\ T_s(t(m)) \le T_c(t(m)) + \Delta$. Тогда запрос m можно считать стабильным для s в момент времени t, если $T_c(t(m)) < T_s(t) - \Delta$.

Следующий алгоритм для упорядочивания клиентских запросов использует *идентификаторы запросов, генерируемые серверами* в отличие от предыдущих алгоритмов, в которых порядок запросов определялся клиентами. Цель данного алгоритма – присвоить каждому клиентскому запросу m уникальный идентификатор $uid(m)$, который становится известным всем исправным серверам. Для этого каждый сервер s предлагает свое значение $cuid(s, m)$, которое рассылает его всем остальным серверам, а в качестве итогового значения берется $uid(m) = \max_{s' \in S_{up}} cuid(s', m)$.

Каждый сервер свой $cuid(s, m)$ формирует следующим образом:

$$cuid(s, m) = \max(\lfloor SEEN_s \rfloor, \lfloor ACCEPT_s \rfloor) + 1 + \frac{1}{n},$$

где $SEEN_s = \max_{m' \in H_s} cuid(s, m')$, $ACCEPT_s = \max_{m' \in H_s} uid(m')$ для тех m', для которых это значение известно.

В таком случае запрос m считается стабильным для s, если $\neg\exists m' : cuid(s, m') \le uid(m)$ и $uid(m')$ еще не вычислен.

Алгоритм консенсуса – это отказоустойчивый способ принятия единого решения в распределенной среде. Формально алгоритм консенсуса определяется следующим образом [7]. Пусть имеется набор процессов $p_1, \ldots, p_N$. У каждого процесса p_i определена переменная d_i, в которой хранится решение данного процесса. Переменная d_i изначально имеет неопределенное значение $null$. Любой процесс p_i может предложить всем остальным процессам принять в качестве решения некоторое значение v_i. Далее происходит обмен сообщениями между процессами с целью принятия единого решения. Работа алгоритма завершается, когда для каждого процесса p_i в его переменной d_i записано одно из значений $v_1, \ldots, v_n$, причем для всех процессов значения, записанные в d_i, должны совпадать.

Для упорядочивания клиентских запросов данный алгоритм можно использовать следующим образом. При получении клиентского запроса m сервер s запускает алгоритм консенсуса, в котором он предлагает принять в качестве решения запрос m. Если в результате работы алгоритма в переменной d_i оказывается значение m, то m считается стабильным.

Различные реализации алгоритма консенсуса описаны в работах [8; 9; 10]. Выбор конкретной реализации является задачей, требующей отдельного рассмотрения.

Репликация с первичной и вторичными копиями

Подход с использованием первичной и вторичных реплик, предложенный в [3] для систем управления баз данных, также может быть применен в рамках ранее описанной модели.

Одна из реплик объявляется как первичная, все остальные – вторичные. Операции над объектом считаются завершенными, как только модифицирована первичная копия. Сервер с первичной репликой считается ответственным за распространение обновлений на все вторичные копии в течение некоторого последующего времени.

Недостатком данного подхода является то, что некоторые операции могут не дойти до вторичных копий при отказе сервера первичной реплики. Данный подход следует применять, только если такая ситуация является приемлемой. Чтобы избавиться от данного минуса, потребовалось бы отказаться от асинхронного распространения обновлений и считать операцию завершенной только после ее применения на всех копиях. Но тогда данный подход ничем не будет выигрывать у state-machine репликации, и в то же время отказ сервера первичной реплики по-прежнему будет требовать проведения восстановления, выбора новой первичной копии.

Оценка алгоритмов репликации

При использовании репликации следует учитывать, что она потребляет дополнительные ресурсы системы. Это накладывает ограничения на использование определенных способов репликации, а также на количество поддерживаемых реплик.

Пусть $|S| = N$. Для алгоритмов репликации определим следующий набор характеристик, по которым их можно сравнивать:

- Количество сообщений, посылаемых алгоритмом репликации, на каждый клиентский запрос – $m(N)$.
- Допустимость возникновения временной несогласованности копий. При выборе алгоритма репликации важно учитывать, возможна ли ситуация, при которой клиент может обнаружить нарушение взаимно однозначного соответствия между соответствующими данными реплик. Если данная ситуация происходит, то нарушается принцип «прозрачности репликации».
- Дополнительная нагрузка, не связанная напрямую с клиентскими запросами.

Произведем расчет данных характеристик для каждого способа репликации, рассмотренного ранее.

Логические часы. Чтобы осуществить запрос, клиент рассылает по одному сообщению каждому серверу. Также для данного метода требуется периодическая рассылка heartbeat сообщений от каждого клиента к каждму серверу. Так что для выполнения одной операции над объектом необходима передача $m(N) = N^2$ сообщений.

Синхронизированные часы реального времени. В данном алгоритме клиенту также требуется рассылка всего одного сообщения каждому серверу. Вместо механизма heartbeat используется задержка по времени Δ, после которой гарантированно не придет запрос, который необходимо исполнить раньше текущего. Это говорит о том, что клиенты не могут делать запросы чаще, чем раз в Δ секунд. Также возникает дополнительная нагрузка, связанная с необходимостью постоянной синхронизации часов.

Идентификаторы, генерируемые серверами. Оценка данного алгоритма проста – каждый процесс должен разослать значение $cuid(r, m)$ всем исправным серверам, т. е. $m(N) = N^2$.

Алгоритм консенсуса. Если рассматривать реализации алгоритма консенсуса с выделенным лидером [9; 10], то в течение исправной работы лидера на каждый запрос рассылается

порядка $4N$ сообщений между лидером и всеми остальными серверами: *PrepareRequest, PrepareResponse, AcceptRequest, AcceptResponse*.

Репликация с первичной и вторичными копиями. В данном алгоритме все модификации, произведенные над первичной копией, должны быть распространены на остальные реплики. Как указано в [3], это можно сделать в лучшем случае посредством $2N + 3$ сообщений.

В промежуток времени между применением операции на первичной копии и распространением обновления на вторичные реплики, данные различных копий могут находиться в несогласованном состоянии.

Все полученные оценки сведены в одну общую таблицу.

Сравнение методов репликации

Способ репликации	$m(N)$	Допустимость временной несогласованности	Примечания
State-machine replication + логические часы	N^2	нет	
State-machine replication + синхронизированные часы реального времени	N	нет	Необходимость постоянной синхронизации часов. Гарантированная минимальная временная задержка Δ перед обработкой запроса
State-machine replication + идентификаторы, генерируемые репликами	N^2	нет	
State-machine replication + алгоритм консенсуса	$4N$	нет	Дополнительные нагрузки при выборе лидера
Первичная и вторичные копии	$2N + 3$	да	Возможны потери данных при отказе первичной копии

Выводы

Репликация позволяет повысить надежность распределенной системы ценой внесения избыточности по данным и накладных расходов на синхронизацию копий. Существует несколько подходов к репликации, отличающихся по производительности и надежности.

Наиболее привлекательным из предложенных является вариант с использованием state-machine replication на основе алгоритма консенсуса, как обеспечивающий стопроцентную согласованность объектов при наименьших накладных расходах.

Список литературы

1. Надежность в технике. Термины и определения [Текст]: *ГОСТ Р 27.002–2009.* Введ. 2009-12-09. М.: Стандартинформ, 2011, 27 с.

2. *Тель Ж.* Введение в распределенные алгоритмы: Пер. с англ. М.: МЦНМО, 2009. 616 с.

3. *Дейт К. Дж.* Введение в системы баз данных. 8-е изд.: Пер. с англ. М.: Изд. дом «Вильямс», 2005. 1328 с.: ил. Парал. тит. англ.

4. *Schneider F. B.* Implementing Fault-Tolerant Sevices Using the State Machine Approach: A Tutorial // ACM Computing Surveys. 1990. Vol. 22. No. 4 (December). P. 299–319.

5. *Lamport L.* Time, clocks and the ordering of events in a distributed system // Communications of the ACM. July 1978. Vol. 21. No. 7. P. 558–564.

6. *Schneider F. B.* A paradigm for reliable clock synchronization // Proc. Advanced Seminar on Real-Time Local Area Networks. April 1986. P. 85–104.

7. Distributed Systems: Concepts and Design / G. Coulouris [et al.]. Reading: Addison-Wesley, 2011 1008 p.

8. *Lamport L.* The part-time parliament. // ACM Transactions on Computer Systems. May 1998. Vol. 16. No. 2 P. 133–69

9. *Ongaro D.* In search of an understandable consensus algorithm / D. Ongaro, J. Ousterhout // Proc. ATC'14 USENIX Annual Technical Conference (June 2014). P. 305–320.

10. *Liskov B.* Viewstamped replication revisited / B. Liskov, J. Cowling // Tech. Rep. MIT-CSAIL-TR-2012-021, 2013.

Материал поступил в редколлегию 17.02.2016

A. V. Il'in [1], **B. N. Pishik** [2]

[1] *Novosibirsk State University*
2 Pirogov Str., Novosibirsk, 630090, Russian Federation

[2] *Design Technological Institute of Digital Techniques of SB RAS*
6 Acad. Rzhanov Str., Novosibirsk, 630090, Russian Federation

sanekspot@gmail.com, boris.pishchik@kti.nsc.ru

REPLICATION METHODS IN DISTRIBUTED SYSTEMS

Discussed various methods of object replication in an unreliable distributed computing environment. Two main approaches is the use of state-machine replication and primary and secondary copies. Described several algorithms implementing state-machine method. The methods are compared on uniform criteria.

Keywords: distributed system, replication, fault tolerance, state-machine, the algorithm of consensus.

References

1. Reliability in technique. Terms and definitions [Text]: GOST R 27.002-2009. Introduction. 2009-12-09. M.: STANDARTINFORM, 2011, p. 27 .

2. Tel G. Introduction to distributed algorithms. TRANS. Eng. M.: MCNMO, 2009. P. 616.

3. Date C. J. Introduction to database systems, 8th edition. : TRANS. Eng. M.: Publishing house "Williams", 2005. P. 1328: ill. Paral. Titus. Eng.

4. Schneider F. B. Implementing Fault-Tolerant Sevices Using the State Machine Approach: A Tutorial // ACM Computing Surveys, Vol. 22, № 4 (December 1990). P. 299–319.

5. Lamport L. Time, clocks and the ordering of events in a distributed system // Communications of the ACM, Vol. 21, № 7 (July 1978). P. 558–564.

6. Schneider F. B. A paradigm for reliable clock synchronization. // Proc. Advanced Seminar on Real-Time Local Area Networks (April 1986). P. 85–104.

7. Distributed Systems: Concepts and Design / G. Coulouris [et al.] - Reading: Addison-Wesley, 2011. 1008 p.

8. Lamport L. The part-time parliament. // ACM Transactions on Computer Systems Vol. 16, № 2 (May 1998). p. 133–169

9. Ongaro D. In search of an understandable consensus algorithm. / D. Ongaro, J. Ousterhout // Proc. ATC'14 USENIX Annual Technical Conference (June 2014). p. 305–320.

10. Liskov B. Viewstamped replication revisited. / B. Liskov, J. Cowling // Tech. Rep. MIT-CSAIL-TR-2012-021, 2013.