УДК 004.42
DOI: 10.15827/0236-235X.124.763-767

Дата подачи статьи: 18.05.18
2018. Т. 31. № 4. С. 763–767

Сравнительный анализ реализаций спин-блокировок

Д.С. Тараканов [1], *аспирант, nl40.000@gmail.com*
М.С. Косяков [1], *к.т.н., доцент, mkosyakov@gmail.com*

[1] *Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, г. Санкт-Петербург, 197101, Россия*

Работа посвящена проблеме производительности многопоточных приложений, использующих спин-блокировки для синхронизации доступа к критической секции. Дано развернутое описание этого механизма с указанием потенциально узких мест. Проведен анализ существующих реализаций спин-блокировок, таких как POSIX-блокировка, билетная спин-блокировка и MCS-блокировка, с детальным разъяснением их внутреннего устройства.

Для оценки эффективности введены две характеристики: оперативность приложения (время в процессорных циклах) и выполнение условия справедливости (определяется количественно). Для сравнения этих реализаций при использовании многоядерного процессора с общей кэш-памятью последнего уровня подготовлены и оптимизированы описанные спин-блокировки. Также разработано тестовое приложение, позволяющее варьировать количество потоков и полезную нагрузку в критической секции и вне ее. По результатам выполнения тестов построены графики и таблицы для значений введенных характеристик.

На основе полученных данных выявлены характерные проблемы (недостаточная масштабируемость, невыполнение условия справедливости) для POSIX-блокировки и билетной спин-блокировки, негативно влияющие на производительность приложений, использующих эти примитивы синхронизации. Проведен подробный анализ причин возникновения данных проблем, который показал, что использование POSIX-блокировки и билетной спин-блокировки обусловливает необходимость постоянной синхронизации кэш-памяти. Дополнительно показано, что POSIX-блокировка не является справедливой, а MCS-блокировка потенциально может быть неэффективной при малом количестве потоков. Полученные результаты могут как использоваться для дальнейших исследований в области примитивов синхронизации, связанных с другими реализациями MCS-механизма или неблокирующими алгоритмами, так и учитываться при выборе спин-блокировки для применения ее в реальном приложении.

Ключевые слова: *многопоточность, многоядерные процессоры, примитивы синхронизации, спин-блокировка.*

Современные операционные системы предоставляют механизм многопоточности [1]. При использовании многопоточности возникает необходимость решения задачи взаимного исключения при доступе потоков к некоторым общим ресурсам, который осуществляется из секции кода, называемой критической.

Для этого применяются различные механизмы взаимного исключения. Один из таких механизмов – спин-блокировки, являющиеся частным случаем одноместного семафора [2, 3]. Их главная особенность состоит в том, что при занятой критической секции поток уходит в так называемый холостой цикл – постоянную проверку значения переменной спин-блокировки, пока она не начнет удовлетворять условию свободной критической секции. Таким образом, спин-блокировка не требует использования системного вызова для своей работы.

Можно выделить основные проблемы, связанные с некоторыми реализациями спин-блокировок: нарушение условия справедливости (нет гарантии, что потоки получают доступ к критической секции в порядке запросов на вход) или низкая масштабируемость (падение производительности при увеличении количества потоков, взаимодействующих с критической секцией). Наличие таких проблем в многопроцессорных системах подтверждается уже проведенными исследованиями [4, 5].

Стоит отметить, что причины названных проблем могут быть специфичны для определенных систем. Например, слабая масштабируемость спин-блокировок на многопроцессорных системах прежде всего связана с синхронизацией кэш-памяти процессоров при освобождении критической секции. Тем не менее, при использовании одного многоядерного процессора общим для разных ядер является только кэш последнего уровня, а значит, если возникнет необходимость доступа к одной и той же области памяти у разных ядер, будет происходить синхронизация кэш-памяти между ядрами. Это является одним из узких мест при использовании кэш-памяти [6]. Влияние данной проблемы синхронизации на масштабируемость спин-блокировок не установлено.

В данной работе для оценки эффективности различных реализаций спин-блокировок и их сравнения между собой использовались следующие характеристики: оперативность приложения (τ – время (в процессорных циклах), необходимое приложению на выполнение заданного количества итераций count) и условия справедливости – условие для спин-блокировки, подразумевающее одинаковое число захватов критической секции каждым из конкурирующих потоков (например, ожидается, что при выполнении четырьмя потоками 1 000 итераций каждый из потоков выполнит по $x_1 = x_2 = x_3 = x_4 = x_{fair} = 250$ итераций). В качестве количе-

ственной характеристики выполнения условия справедливости в работе рассматривалось среднее отклонение от x_{fair}, вычисляемое по формуле $\delta = \frac{1}{N}\sum_{k=1}^{N}\frac{\left|x_k - x_{fair}\right|}{x_{fair}}100\,\%$, где $k \in 1, \ldots, N$, N – число конкурирующих потоков. Данные характеристики были выбраны как наиболее информативные на основе уже проведенных исследований в области определения производительности многопоточных приложений [4, 7, 8].

С целью увеличения производительности многопоточных приложений в данном исследовании были рассмотрены такие реализации спин-блокировок, как POSIX-блокировка, билетная спин-блокировка и MCS-блокировка [4].

Реализации спин-блокировок

POSIX-блокировка представляет собой целочисленную переменную. В разблокированном состоянии она хранит значение 1. Для ее захвата используются операции декремента и проверки флага статуса ZF (zero flag) [6]. Если флаг установлен, то можно заходить в критическую секцию. В противном случае захват спин-блокировки переходит в ожидание освобождения критической секции, используя холостой цикл. К преимуществу такой реализации следует отнести ее предельную простоту, к явному недостатку – невыполнение условия справедливости.

Билетная спин-блокировка является структурой данных, состоящей из двух полей. Первое – номер следующего доступного билета. При попытке захвата такой спин-блокировки поток атомарно увеличивает значение первого поля – номера следующего доступного билета, и сохраняет предыдущее. Затем проверяет значение второго поля – номера обслуживаемого билета. Если эти значения равны, то поток может войти в критическую секцию. В противном случае он уходит в холостой цикл проверки этого поля [9]. Для выхода из критической секции достаточно инкрементировать номер обслуживаемого билета. Данная реализация спин-блокировки гарантирует справедливость, но потенциально может вызывать проблемы с масштабируемостью.

MCS-блокировка (в названии используется аббревиатура из фамилий авторов идеи) также использует очередь в пространстве пользователя. Но здесь она реализуется с помощью связного списка, при этом каждый узел этого списка хранит целочисленную переменную. При попытке захвата такой спин-блокировки поток добавляет себя в очередь. Если она пуста, он заходит в критическую секцию. Иначе поток устанавливает значение своей переменной в единицу и в холостом цикле ожидает ее изменения. При выходе из критической секции поток удаляет себя из очереди, если он единственный в ней, либо же, воспользовавшись тем, что очередь представлена связным списком, изменяет переменную холостого цикла в следующем в очереди потоке, позволяя ему войти в критическую секцию [10].

Использование связного списка и разделение между потоками переменных холостого цикла позволяют гарантировать этой спин-блокировке высокую масштабируемость. В то же время это приводит к усложнению ее реализации и дополнительным накладным расходам.

Параметры проведения эксперимента

При непосредственном проведении экспериментов использовалось следующее оборудование: компьютер с процессором Intel Core i7-2630QM 2,0 ГГц и отключенной технологией Turbo Boost. В данном процессоре установлены 4 ядра с поддержкой технологии Hyper-threading (позволяет одновременно выполнять до 8 потоков). Кэш первого уровня – 256 Кб, второго – 1 Мб, третьего – 6 Мб.

Для проверки проблем масштабируемости и нарушения условия справедливости при использовании многоядерного процессора были подготовлены и оптимизированы рассмотренные выше реализации спин-блокировок. Они были выложены в свободный доступ [11]. Также было разработано тестовое приложение. Каждый поток выполнял функцию, общий вид которой представлен в листинге:

```
void thread_function(){
        while(true){
                spin_lock();
                factorial(i);
                count--;
                spin_unlock();
                factorial(j);
                if (count <= 0) return;
        }
}
```

В качестве полезной нагрузки, выполнявшейся в критической секции и в промежутках между попытками ее захвата, использовалось вычисление различных значений факториала (*factorial*(*i*) и *factorial*(*j*), где *i*, *j* – параметры теста). Таким образом, в зависимости от параметров *i* и *j* варьировалось соотношение δT времени, проводимого потоком в критической секции (T_{cs}) и вне ее (T_{out}). Оно вычислялось по формуле $\delta T = \frac{T_{cs}}{T_{cs} + T_{out}}100\,\%$. Также варьировалось количество потоков N, используемых приложением.

Сравнительный анализ спин-блокировок

По результатам тестирования были получены следующие результаты. На рисунках отмечен раз-

брос результатов измерения времени с доверительной вероятностью 95 %. Стоит отметить, что для MCS-блокировки и билетной спин-блокировки он составил доли процентов от измеренных результатов и не виден в масштабе рисунка.

Для приложения, в котором потоки выполняют более длительную задачу в критической секции, а не вне ее (85 % времени выполнения приходилось на критическую секцию), при большом количестве потоков лучшую эффективность продемонстрировала MCS-блокировка (рис. 1). Схожую эффективность имела и билетная спин-блокировка.

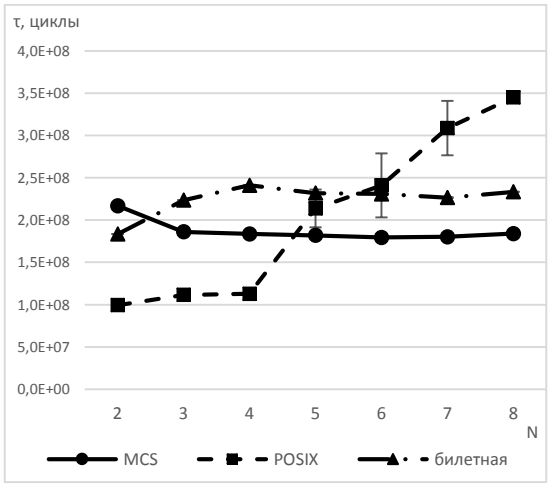

Рис. 1. Зависимость времени τ от количества потоков N при δT = 85 %

Fig. 1. The dependence of time τ on the number of threads N at δT = 85 %

Также стоит отметить лучшее время работы POSIX-блокировки при небольшом количестве потоков. Но, как можно увидеть в таблице 1, это связано с тем, что она не гарантировала доступность критической секции для всех потоков, то есть некоторые потоки ни разу не смогли получить доступ к критической секции. Незначительные отклонения от 0 % (для билетной спин-блокировки и MCS-блокировки) связаны с неравномерным стартом и прекращением работы потоков.

Таблица 1

Отклонение количества захватов критической секции δ (%) при δ*T* = 85 %

Table 1

Deviation of the number of captures of a critical section δ (%) at δ*T* = 85 %

Блокировка	Количество потоков						
	2	3	4	5	6	7	8
MCS	0,04	0,01	0,09	0,1	4,1	1	1,7
POSIX	100	120	120	102	62	30	25
Билетная	0,03	0,4	0,07	0,08	0,2	0,03	0,1

Дополнительно это приводит к большому разбросу результатов измерений для POSIX-блокировки – в разных тестовых запусках порядок доступа потоков к критической секции мог сильно отличаться, что влияло на необходимость в синхронизации кэш-памяти между ядрами процессора.

Для приложения с δ*T* = 8 % наблюдается схожее поведение (рис. 2). Стоит отметить понижение эффективности для билетной спин-блокировки и MCS-блокировки при трех потоках. Это связано с возникающей конкуренцией за критическую секцию и очередью на вход в нее.

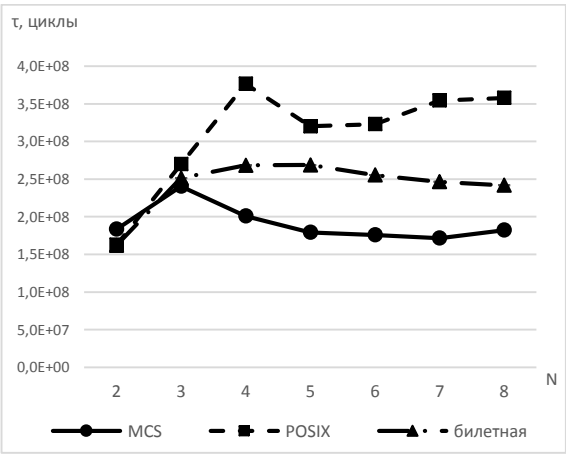

Рис. 2. Зависимость времени τ от количества потоков N при δT = 8 %

Fig. 2. The dependence of time τ on the number of threads N at δT = 8 %

При уменьшении конкуренции потоков за критическую секцию все потоки имеют возможность доступа к критической секции. Это можно увидеть в таблице 2 для POSIX-блокировки. Тем не менее, с увеличением количества потоков (следовательно, и с увеличением конкуренции) POSIX-блокировка перестает быть справедливой.

Таблица 2

Отклонение количества захватов критической секции δ (%) при δ*T* = 8 %

Table 2

Deviation of the number of captures of a critical section δ (%) at δ*T* = 8 %

Блокировка	Количество потоков						
	2	3	4	5	6	7	8
MCS	0,01	0,2	0,02	0,8	2,3	0,9	0,4
POSIX	0	0,1	8	28	17	16	12
Билетная	0	0,01	0,04	0,3	0,2	0,05	0,03

POSIX-блокировка показывает плохую масштабируемость, причиной которой является использование декремента переменной холостого цикла при каждой проверке доступности критической секции. Это приводит к необходимости постоянной синхронизации кэш-памяти между ядрами.

Использование билетной спин-блокировки также связано с общей переменной для холостого

цикла. Ее изменение при выходе из критической секции заставляет все ядра процессора сразу синхронизировать кэш-память, что приводит к более низкой эффективности по сравнению с MCS-блокировкой, которая лишена этих недостатков. Тем не менее, в некоторых ситуациях (например, при небольшом количестве потоков) накладные расходы от ее использования могут негативно сказаться на эффективности.

Также стоит отметить, что для δT = 0,5 % наблюдалось повышение эффективности для всех спин-блокировок, что связано с отсутствием очереди при ожидании критической секции. При таком характере работы с критической секцией вид спин-блокировки не влияет на доступ потоков к ней (рис. 3 и табл. 3).

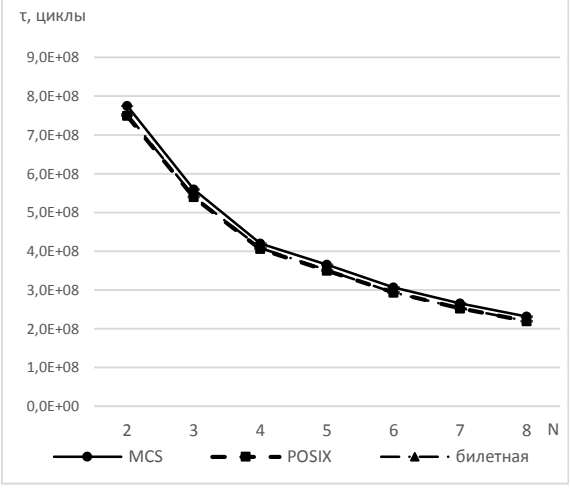

Рис. 3. Зависимость времени τ от количества потоков N при δT = 0,5 %

Fig. 3. Dependence of time τ on number of threads N at δT = 0,5 %

Таблица 3

Отклонение количества захватов критической секции δ (%) при δT = 0,5 %

Table 3

Deviation of the number of captures of a critical section δ (%) at δT = 0,5 %

Блокировка	Количество потоков						
	2	3	4	5	6	7	8
MCS	0	0	0,1	0,6	3	0,3	0,6
POSIX	0,01	0,02	0,04	0,5	0,07	0,03	0,05
Билетная	0,02	0	0,02	2,4	1,6	0,6	0,03

Заключение

В работе рассмотрены различные способы реализации спин-блокировок для многопоточных приложений. Для сравнения их эффективности при использовании многоядерного процессора было разработано тестовое приложение [11].

На основе полученных данных можно сделать вывод о том, что POSIX-блокировка показывает недостаточную масштабируемость в приложениях с большим количеством потоков даже на многоядерных системах с общим кэшем последнего уровня. В то же время билетная спин-блокировка и MCS-блокировка являются более масштабируемыми, но при этом MCS-блокировка может быть неэффективной при малом количестве потоков. Основные проблемы масштабируемости связаны с синхронизацией кэш-памяти между ядрами процессора.

Дальнейшие исследования могут быть связаны со сравнением вышеназванных спин-блокировок с альтернативными реализациями MCS-механизма, такими как CLH-блокировка [12] или K42-блокировка [13], или с неблокирующими алгоритмами синхронизации [14].

Литература

1. Столлингс В. Операционные системы. М.: Вильямс, 2013. 848 с.

2. Dijkstra E.W. Solution of a problem in concurrent programming control. Communications of the ACM, 1983, vol. 26, no. 1, pp. 21–22. DOI: 10.1145/357980.357989.

3. Dijkstra E.W. Cooperating sequential processe. Reprinted Programming Languages, F. Genuys Ed., NY, Academic Press, 1968, pp. 43–112. DOI: 10.1007/978-1-4757-3472-0_16.

4. Boyd-Wickizer S., Kaashoek M.F., Morris R., Zeldovich N. Non-scalable locks are dangerous. Proc. of the Linux Sympos., 2012, pp. 119–130.

5. Anderson T.E. The performance of spin lock alternatives for shared-money multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 1990, vol. 1, no. 1, pp. 6–16. DOI: 10.1109/71.80120.

6. Intel 64 and IA-32 Architectures Optimization Reference Manual. URL: https://filepursuit.com/directory/3507273-Intel-64-and-IA-32-Architectures-Optimization-Reference-Manual-pdf/ (дата обращения: 08.01.2018).

7. Tallent N.R., Mellor-Crummey J.M., Porterfield A. Analyzing lock contention in multithreaded applications. ACM SIGPLAN Notices, 2010, vol. 45, no. 5, pp. 269–279. DOI: 10.1145/1837853.1693489.

8. Adhianto L., Banerjee S., Fagan M., Krentel M., Marin G., Mellor-Crummey J.M., Tallent N.R. HPCToolkit: Tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience, 2010, vol. 22, no. 6, pp. 685–701. DOI: 10.1002/cpe.v22:6.

9. Scott M.L., Scherer W.N. Scalable queue-based spin locks with timeout. ACM SIGPLAN Notices, 2001, vol. 36, no. 7, pp. 44–52.

10. Mellor-Crummey J.M., Scott M.L. Algorithms for scalable synchronization on shared-memory multiprocessors. ACM TOCS, 1991, vol. 9, no. 1, pp. 21–65. DOI: 10.1145/103727.103729.

11. Liblock. 2017. URL: https://github.com/NaglijLamer/liblock (дата обращения: 08.01.2018).

12. Luchangco V., Nussbaum D., Shavit N. A hierarchical CLH queue lock. LNCS, 2006, vol. 4128, pp. 801–810. DOI 10.1007/11823285_84.

13. Appavoo J., Auslander M., Butrico M., da Silva D.M., Krieger O., Mergen M.F., Ostrowski M., Rosenburg B., Wisniewski R.W., Xenidis J. Experience with K42, an open-source, Linux-compatible, scalable operating-system kernel. IBM Systems J., 2005, vol. 44, no. 2, pp. 427–440. DOI: 10.1147/sj.442.0427.

14. Michael M.M., Scott M.L. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. Proc. of the Annual ACM Sympos. on Principles of Distributed Comp., 1996, pp. 267–275. DOI: 10.1145/248052.248106.

Software & Systems
DOI: 10.15827/0236-235X.124.763-767

Received 18.05.18
2018, vol. 31, no. 4, pp. 763–767

Comparative analysis of spinlock realizations

D.S. Tarakanov [1], *Postgraduate Student, nl40.000@gmail.com*
M.S. Kosyakov [1], *Ph.D. (Engineering), Associate Professor, mkosyakov@gmail.com*

[1] *The National Research University of Information Technologies, Mechanics and Optics, St. Petersburg, 197101, Russian Federation*

Abstract. The paper considers the problem of performance of multithreaded applications using spinlocks to synchronize access to a critical section. There is a detailed description of this mechanism, indicating potential bottlenecks. There is also an analysis of the existing spinlock implementations (such as POSIX spinlock, ticket spinlock and MCS spinlock) with a detailed explanation of their internal structure.

In order to evaluate the effectiveness, the authors introduce two characteristics as follows: application agility (time in processor cycles) and the fulfillment of a fairness condition (quantified). The described spinlocks have been prepared and optimized to compare these implementations of spinlocks using a multi-core processor with shared last level cache. In addition, a test application is developed that allows varying the number of threads and the payload in the critical section and outside. The paper contains diagrams and graphs for the values of the introduced charachteristics.

Based on the data obtained, characteristic problems (insufficient scalability, non-fulfillment of the fairness condition) for POSIX-spinlock and ticket spinlock are revealed, which negatively affect the performance of applications using these synchronization primitives. An analysis of the causes of these problems shows that the use of POSIX-spinlock and ticket spinlock leads to the need for permanent cache synchronization. Additionally, it is shown that POSIX-spinlock is not fair, and MCS-spinlock might potentially be inefficient with a small number of threads.

The obtained results might be used both for further research of synchronization primitives associated with other implementations of the MCS-mechanism or non-blocking algorithms and be considered when choosing a spinlock to use in a real application.

Keywords: multithreading, multicore processors, synchronization primitives, spinlock.

References

1. Stollings V. *Operation Systems*. Moscow, Vilyams Publ., 2013, 848 p.

2. Dijkstra E.W. Solution of a problem in concurrent programming control. *Comm. of the ACM*. 1983, vol. 26, no. 1, pp. 21–22. DOI: 10.1145/357980.357989.

3. Dijkstra E.W. *Co-operating Sequential Processes*. *Programming Language*. F. Genuys (Ed.), NY, Academic Press, 1968, pp. 43–112. DOI: 10.1007/978-1-4757-3472-0_16.

4. Boyd-Wickizer S., Kaashoek M.F., Morris R., Zeldovich N. Non-scalable locks are dangerous. *Proc. Linux Symp*. 2012, pp. 119–130.

5. Anderson T.E. The performance of spin lock alternatives for shared-money multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*. 1990, vol. 1, no. 1, pp. 6–16. DOI: 10.1109/71.80120.

6. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Available at: https://filepursuit.com/directory/3507273-Intel-64-and-IA-32-Architectures-Optimization-Reference-Manual-pdf/ (accessed January 8th, 2018).

7. Tallent N.R., Mellor-Crummey J.M., Porterfield A. Analyzing lock contention in multithreaded applications. *ACM SIGPLAN Notices*. 2010, vol. 45, no. 5, pp. 269–279. DOI: 10.1145/1837853.1693489.

8. Adhianto L., Banerjee S., Fagan M., Krentel M., Marin G., Mellor-Crummey J.M., Tallent N.R. HPCToolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*. 2010, vol. 22, no. 6, pp. 685–701. DOI: 10.1002/cpe.v22:6.

9. Scott M.L., Scherer W.N. Scalable queue-based spin locks with timeout. *SIGPLAN Notices (ACM Special Interest Group on Programming Languages)*. 2001, vol. 36, no. 7, pp. 44–52.

10. Mellor-Crummey J.M., Scott M.L. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans. on Computer Systems (TOCS)*. 1991, vol. 9, no. 1, pp. 21–65. DOI: 10.1145/103727.103729.

11. *Liblock*. Available at: https://github.com/NaglijLamer/liblock (accessed January 8, 2018).

12. Luchangco V., Nussbaum D., Shavit N. A hierarchical CLH queue lock. *Lecture Notes in Computer Science*. 2006, vol. 4128, pp. 801–810. DOI 10.1007/11823285_84.

13. Appavoo J., Auslander M., Butrico M., da Silva D.M., Krieger O., Mergen M.F., Ostrowski M., Rosenburg B., Wisniewski R.W., Xenidis J. Experience with K42, an open-source, Linux-compatible, scalable operating-system kernel. *IBM Systems J*. 2005, vol. 44, no. 2, pp. 427–440. DOI: 10.1147/sj.442.0427.

14. Michael M.M., Scott M.L. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. *Proc. Annual ACM Symp. on Principles of Distributed Computing*. 1996, pp. 267–275. DOI: 10.1145/248052.248106.