Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра системного программирования

Сравнение распределителей памяти для многопоточного обработчика транзакций

Курсовая работа

Выполнил: Чередник К.Е.
Научный руководитель: Смирнов К.К.

Санкт-Петербург 2012

Оглавление

Введение

Работа с динамической памятью является важной задачей при разработке приложений. Как показывает практика [6], распределитель памяти, используемый в стандартной библиотеке glibc, демонстрирует плохую производительность на конкретных классах задач — это неизбежная плата за его универсальность. Поэтому в тех случаях, где работа с памятью особенно важна, разработчики вынуждены использовать альтернативные решения, пригодные для каждого конкретного приложения.

В данной работе делается краткий обзор существующих библиотек, разбирается использование памяти многопоточным обработчиком транзакций, проводится экспериментальное сравнение производительности системы при использовании различных распределителей.

Описание системы

В качестве изучаемой системы использовалась система многомерного индексирования в оперативной памяти, разработанная в рамках соревнования при конференции ACM SIGMOD 2012. Система была реализована как библиотека на языке C++, динамически связываемая с тестовой программой организаторов.

Система работает в два этапа: Построение индексов, где заполнение ведётся одной клиентской нитью и интересующим нас показателем является время работы. Поиск по индексам, где запросы поступают от многих нитей, и интересующим нас показателем является количество обработанных запросов за единицу времени. В качестве основных структур данных, используемых для организации индексов, использовались R-деревья и B-деревья. Поддержка одновременного доступа к индексам из разных нитей была реализована с использованием структуры GiST.

Схема работы системы представлена на рисунке 1.

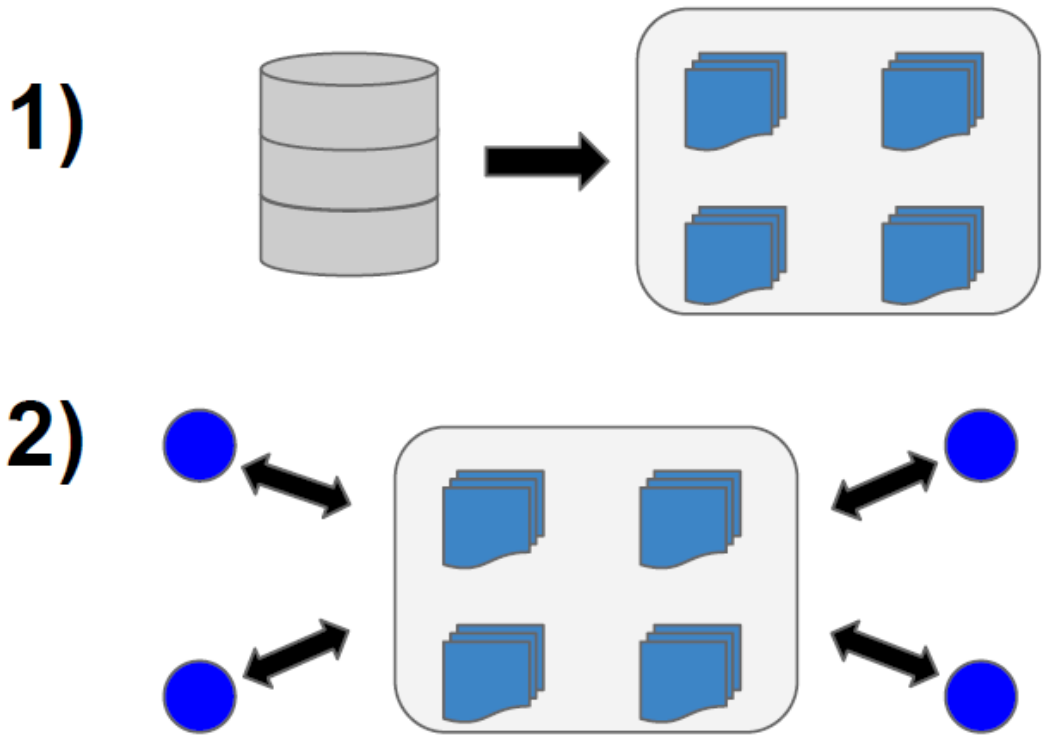

Рисунок 1: Описание работы системы

Обзор готовых технических решений

В настоящее время рынок программного обеспечения располагает некоторым выбором библиотек, реализующих различные алгоритмы распределения памяти. Ниже приведен краткий обзор подобных свободно распространяемых программных средств.

1. Стандартная библиотека Linux glibc использует улучшенную реализацию ptmalloc [3] в качестве многопоточного распределителя памяти. В свою очередь, ptmalloc разрабатывался на основе dl-malloc (Doug Lee Malloc) с улучшенной поддержкой многопоточности.
2. Разрабатываемый компанией Google tcmalloc (Thread-Caching Malloc) [1] демонстрирует лучшую, чем у ptmalloc, производительность при работе в многопоточной среде. В частности, в некоторых работах [2] утверждается, что использование tcmalloc вместо стандартного распределителя памяти позволяет увеличить пропускную способность транзакционной системы вдвое.
3. Jemalloc [5] используется в *BSD как системный распределитель памяти. Также является распределителем в проектах Mozilla Firefox и Facebook.
4. Hoard malloc [4] — еще один кросс-платформенный распределитель памяти, предназначенный для многопоточных приложений в многопроцессорной среде. Является классическим примером альтернативного распределителя памяти при проведении различных замеров.

Для обоснованного выбора распределителя памяти необходимо изучить то, как системой используется память: какого размера блоки и с какой частотой выделяются.

Предварительно можно сформулировать две гипотезы относительно потребления памяти нашей системой:

1. Ожидается выделение большого числа блоков фиксированного размера при небольшой вариации самих размеров. Это обусловлено наличием большого числа блоков, хранящих в себе данные поисковых ключей объектов и хранящих сами объекты фиксированных размеров.
2. Различные нити должны демонстрировать схожее потребление динамической памяти. Это следствие их равноправности.

Экспериментальная часть

Эксперименты проводились в два этапа: получение журнала использования памяти и сравнение различных распределителей.

Использовалась следующая программно-аппаратная конфигурация:

HW: Intel(R) Pentium(R) D CPU 3.00GHz, 3Gb RAM

SW: OS GNU/Linux, kernel 2.6.38.7

Строились 4 многомерных индекса с целочисленными координатами. Характеристики индексов приведены в Таблице 1.

Размер индекса (в Mb)	Размерность индекса	Полезные данные (в байтах)
2	3	8
1	1	8
2	4	64
4	8	64

Таблица 1: Описание используемых индексов

Данные строились на двух различных типах нагрузки транзакционной системы:

- построение индексов;
- поиск по индексу, редкие обновления индекса.

Анализ журналов использования памяти

Зависимость количества выделяемых блоков от их размера, полученная при построении индекса, изображена на диаграмме 1:

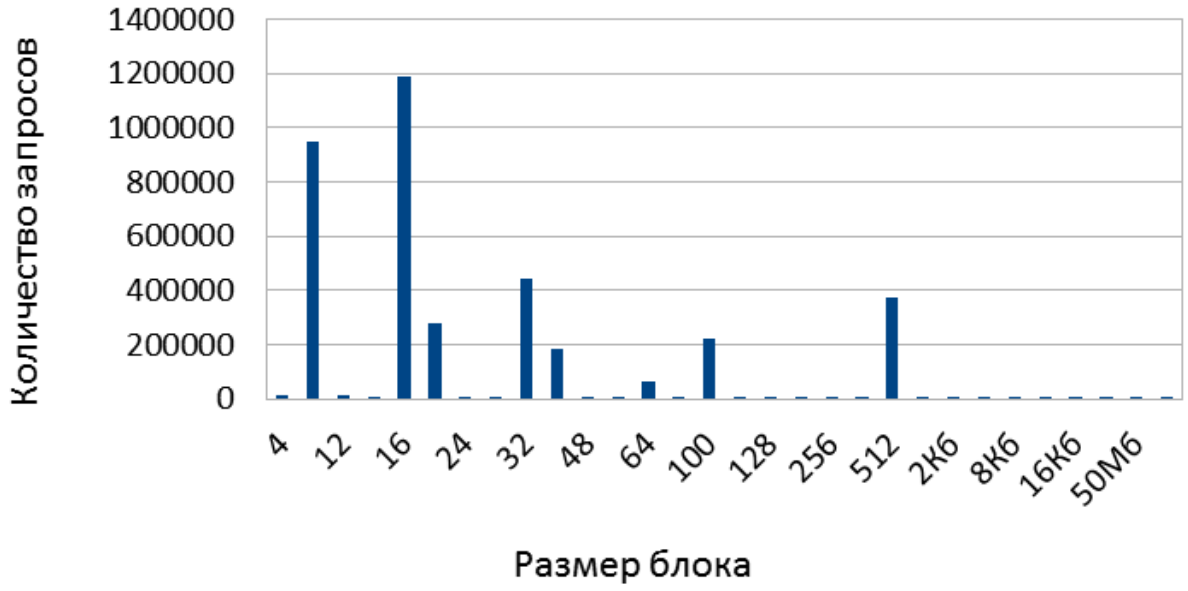

Диаграмма 1: выделение блоков при построении индексов

Как видно на диаграмме, существенное количество запросов на выделение памяти происходит на блоки, размеры которых принадлежат довольно малому множеству (8 элементов).

Более ярко выраженная ситуация проявляется на этапе поиска по индексу (диаграмма 2):

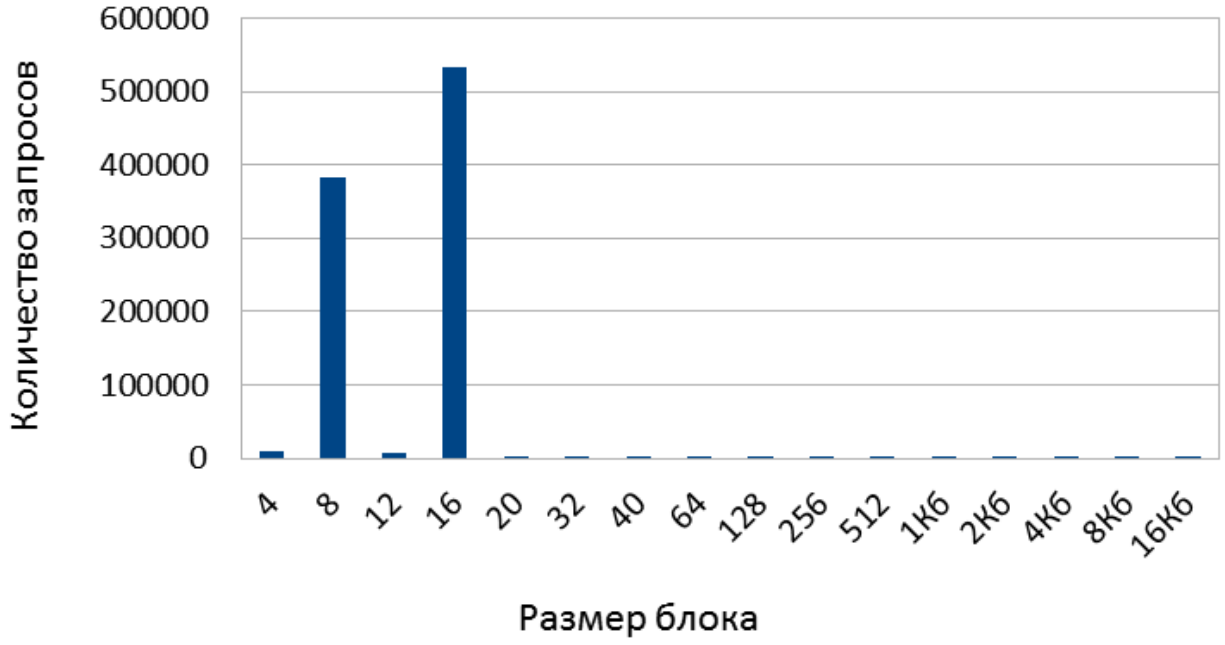

Диаграмма 2: выделение блоков при построении индексов

Таким образом, экспериментально подтверждается первая гипотеза о распределении количества выделяемых блоков по размерам этих блоков.

Количества блоков, выделенных каждой нитью в отдельности, имеет распределение, схожее с изображенным на диаграмме 2. Это подтверждает вторую гипотезу о распределении выделенных блоков между нитями.

Сравнение распределителей памяти

Распределитель	Версия
Glibc	2.14.1
Tcmalloc	2.0
Hoard	38
Jemalloc	2.2.5

Таблица 2: Версии рассматриваемых распределителей

Был проведен ряд экспериментов, в ходе которых была измерена пропускная способность системы в зависимости от количества клиентских нитей и используемого распределителя памяти. Версии распределителей приведены в таблице 2. Для каждого из четырех распределителей памяти и количества нитей от 1 до 20 было проведено 10 замеров, по которым были вычислены итоговые значения с доверительным интервалом 95%. На диаграмме 3 изображена производительность системы на этапе построения индексов.

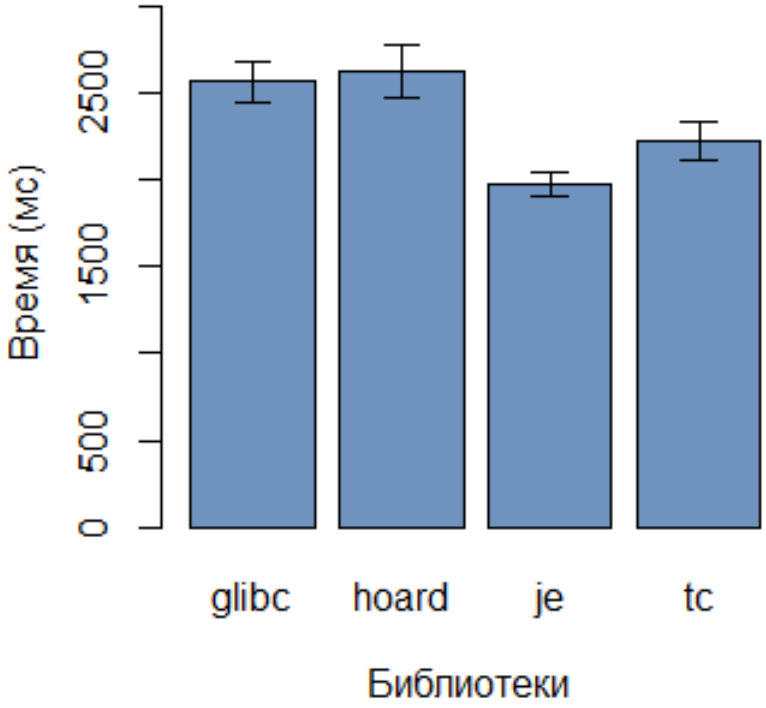

Диаграмма 3: построение индекса

Разница между распределителями glibc, hoard статистически незначима, jemalloc демонстрирует лучшую производительность, tcmalloc занимает промежуточную позицию. При этом разброс значений для jemalloc меньше, чем у остальных.

Диаграммы, изображающие пропускную способность системы, приведены в приложении 1.

Основываясь на этих измерениях, можно предположить, что при увеличении числа нитей пропускная способность системы, при использовании конкретного распределителя стабилизируется. Для подтверждения этого построим линейную регрессию с функцией $y = A + B \cdot x$, где x – количество клиентских нитей, y – пропускная способность системы (в 1000 оп/с).

Allocator	A	B	σA	σB
Glibc	61.04	-0.10	1.48	0.12
Hoard	49.30	-0.02	1.13	0.09
Jemalloc	55.58	0.11	1.23	0.10
Tcmalloc	49.61	0.13	1.34	0.11

Таблица 3: коэффициенты регрессии

Анализ результатов

По данным отдельных распределителей:

1. Две нити дают почти 100% прирост производительности по сравнению с одной нитью. Это объясняется тем, что система двухъядерная.

2. Добавление еще одной нити также дает некоторый прирост, т.к. в случае блокировки одной нити, процессор выполняет другую нить, а не простаивает.
3. Стабилизация пропускной способности системы с ростом числа нитей, начиная с трех. Это можно объяснить полной занятостью обоих ядер. Более точно можно судить об этом по результатам линейной регрессии.

По данным линейной регрессии:

1. Действительно, имеет место стабилизация пропускной способности (значение 0 для наклона прямой согласуется с полученными результатами).
2. Glibc показал лучшую производительность.
3. Hoard и Tcmalloc отстают и разница между ними статистически незначима.
4. Jemalloc занимает промежуточную позицию.

Такие результаты можно частично объяснить следующим образом:

В данной ситуации можно предположить, что распределитель из Glibc показывает лучшую производительность за счёт малого числа ядер, так как его конкуренты (jemalloc и tcmalloc) нацелены на работу с их большим числом.

Разработка распределителя hoard была приостановлена некоторое время назад, поэтому он не использует некоторые современные приемы. Этим можно объяснить его не очень хорошие показатели. Тем не менее, он был включен в наши эксперименты по причине частого его фигурирования в различных обзорах [6].

Распределитель jemalloc использует основные успешные идеи, примененные в некоторых других популярных распределителях. Это объясняет как достигнутую производительность при интенсивном обращении к памяти во время первой фазы работы системы, так и его неплохое поведение на этапе поиска по индексу.

Распределитель tcmalloc проигрывает jemalloc по той причине, что в нашем приложении нити создаются единожды и выполняются в течение всего времени работы приложения, а в такой ситуации jemalloc показывает лучшие результаты, чем tcmalloc.

Заключение

Различные этапы работы транзакционной системы характеризуются различным использованием динамической памяти. Было экспериментально показано, что те распределители памяти, которые демонстрируют лучшие результаты при построении индексов, могут проигрывать в производительности стандартной реализации glibc при выполнении операций поиска.

Из проведенных экспериментов также можно сделать вывод, что конкретно к данной системе в качестве распределителя памяти лучше всего подходит стандартная библиотека.

Литература

1. Sanjay Ghemawat, Paul Menage. TCMalloc: Thread-Cashing Malloc, http://goog-perftools.sourceforge.net/doc/tcmalloc.html [дата просмотра: 17.04.2012]
2. Mark Callaghan. High Availability MySQL: Double sysbench throughput with TCMalloc, http://mysqlha.blogspot.com/2009/01/double-sysbench-throughput-with_18.html [дата просмотра: 17.04.2012].
3. Wolfram Gloger. Ptmalloc3 — a multi-thread malloc implementation, June 2006. http://svn.netlabs.org/repos/qt4/branches/4.5.1/src/3rdparty/ptmalloc/README [дата просмотра: 17.04.2012]
4. Emery D. Berger, Kathryn S. McKinley, Robert D. Blumofe, and Paul R. Wilson. Hoard: A Scalable Memory Allocator for Multithreaded Applications. ACM SIGPLAN Notices Volume 35 Issue 11, Nov. 2000, NY, USA
5. Jason Evans. A Scalable Concurrent malloc(3) Implementation for FreeBSD, Proceedings of BSDCan 2006, http://www.bsdcan.org/2006/papers/jemalloc.pdf [дата просмотра: 17.04.2012]
6. Joseph Attardi, Neelakanth Nadgir. A Comparison of Memory Allocators in Multiprocessors, 2003. http://developers.sun.com/solaris/articles/multiproc/multiproc.html [дата просмотра: 17.04.2012]
7. Steven Fuerst. Benchmarks of the Lockless Memory Allocator [дата просмотра: 10.05.2012]
   http://locklessinc.com/benchmarks_allocator.shtml

Приложение 1. Графики пропускной способности системы на этапе поиска по индексам

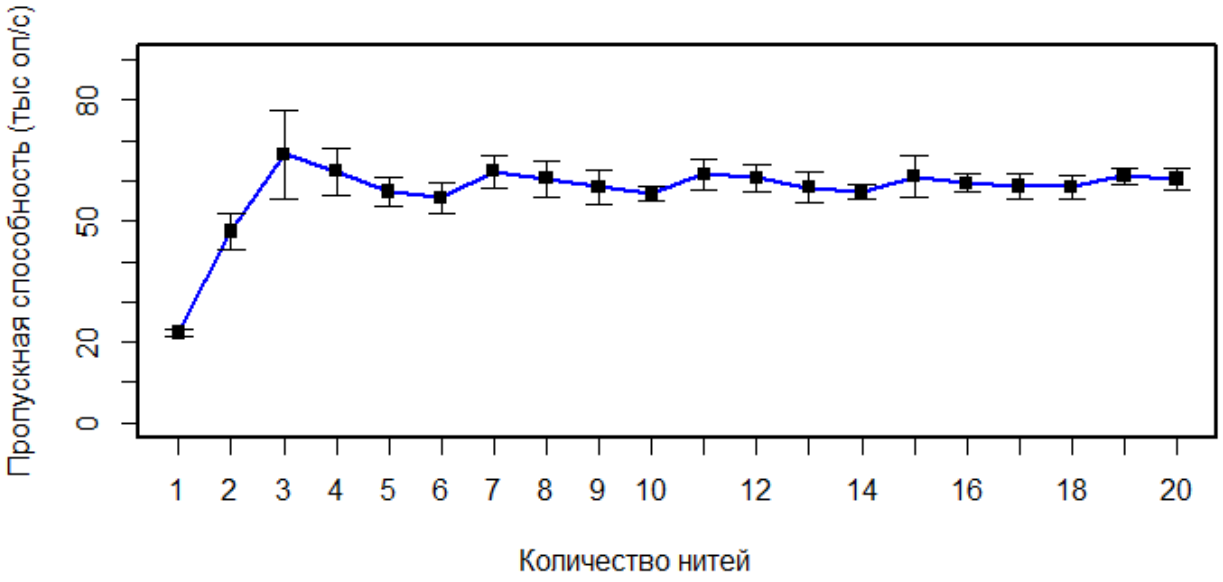

Диаграмма 4: Glibc

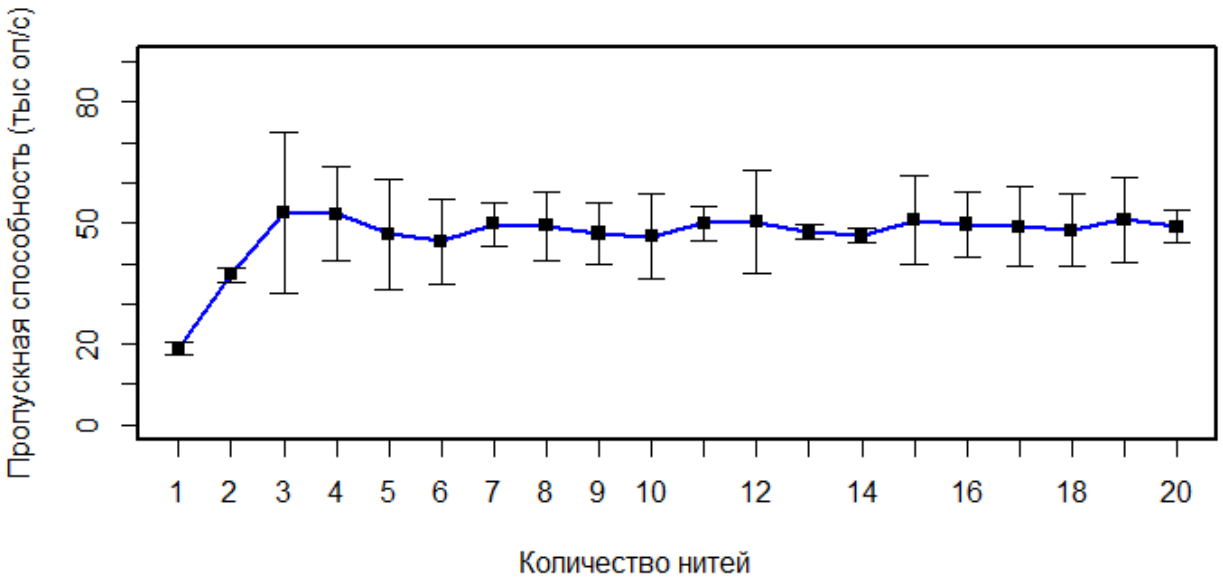

Диаграмма 5: Hoard

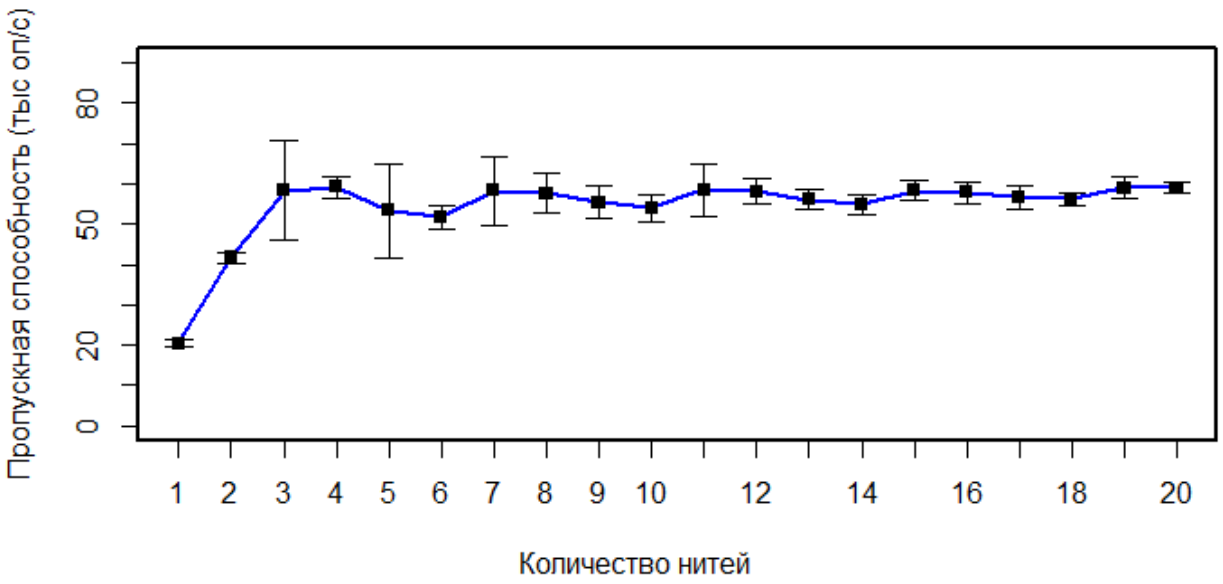

Диаграмма 6: Jemalloc

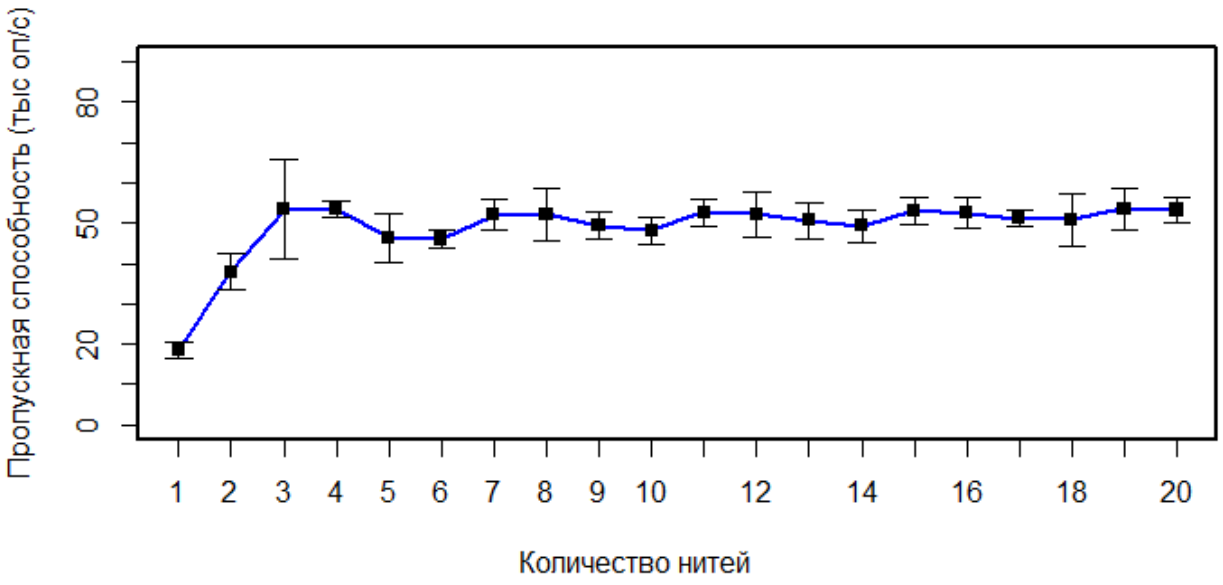

Диаграмма 7: Tcmalloc