**Построение и тестирование token-ring.**

В ходе работы я реализовал протокол токен ринг, собрал статистику с узлов в ходе работы системы, построил графики для latency и throughput при различных параметрах token-ring.

**Описание протокола Token Ring**

1. Система состоит из *N* пронумерованных от *0* до *N-1* узлов (потоков). Узлы упорядочены по порядковому номеру. После состояния *N-1* следует узел *0*, т.е. узлы формируют кольцо.
2. Соседние в кольце потоки могут обмениваться пакетами. Обмен возможен только по часовой стрелке.
3. Каждый поток, получив пакет от предыдущего, отдает его следующему.
4. Пакеты не могут обгонять друг друга.

**Общее описание реализации.**

Система состоит из класса процессора, отвечающего за инициализацию узлов, запуск/остановку процесса пересылки пакетов и сбор статистики.

Класс узла использует блокирующую очередь в которую поступают сообщения от предыдущего узла. Получив сообщение, узел обновляет статистические данные (количество обработанных сообщений, время задержки) и отправляет его в очередь следующего узла.

Main класс запускает и останавливает спустя некоторое время систему, передавая на вход разные параметры числа узлов и числа пакетов, изначально находящихся на каждом узле. После запуска процессора, Main создает ExecutorService для сбора статистики: раз в 10 секунд статистика системы собирается и записывается в csv файл. Процессор получает команду остановить работу, спустя 3 минуты.

**Характеристики.**

1. Latency рассчитывается так: когда узел достает из очереди очередной пакет, он рассчитывает сколько времени прошло с момента transitionStartTime, сохраненного в пакете и прибавляет разницу к переменной totalLatency. После чего сеттит текцщее время в пакет и отправляет в очередь следующего узла. При этом увеличивается число пакетов, отправленных данным узлом. Средняя задержка считается как сумма totalLatency со всех узлов, поделенная на сумму пакетов, отправленных всеми узлами.
2. Throughput считается как усредненное по узлам количество отправленных пакетов, деленое на время, прошедшее со старта системы.

**Варианты реализации очереди**

В ходе экспериментов были рассмотрены 2 варианта очереди: ArrayBlockingQueue и LinkedBlockingQueue.

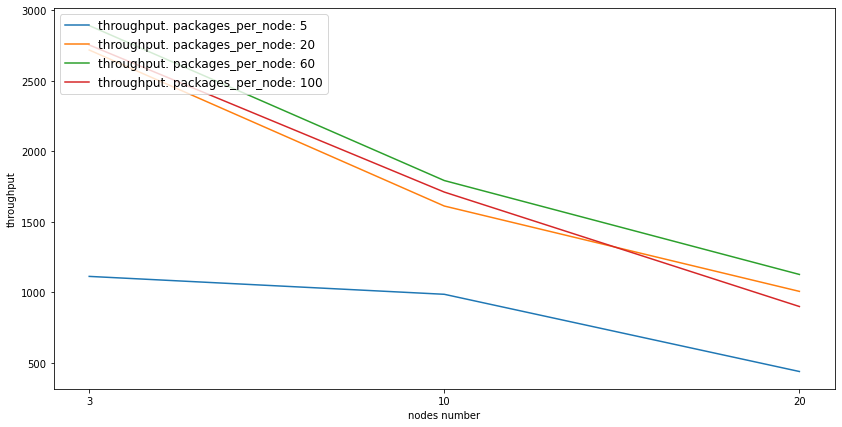
В работе использовался ноутбук с процессором 2,6 GHz 6‑ядерный процессор Intel Core i7.

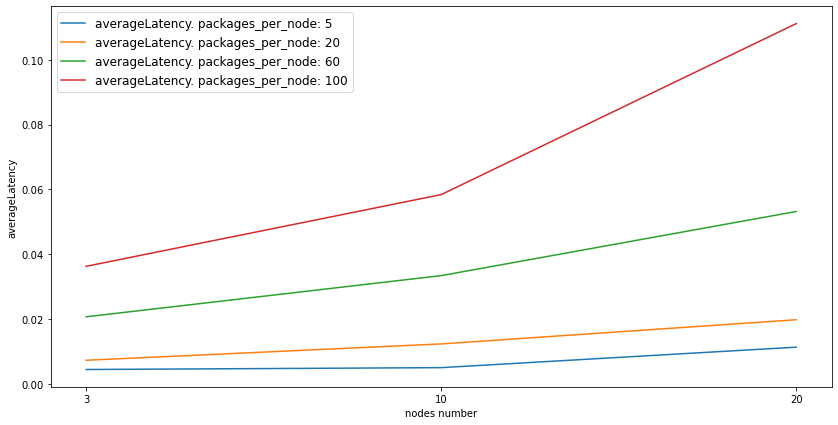
Для ArrayBlockingQueue имеется ограничение на число пакетов, так как размер очереди фиксированный. В реализации если очередь переполнялась, пакеты терялись и увеличивался счетчик потерянных пакетов.

**Графики и статистика (LinkedBlockingQueue)**

Throughput указан в размерности [1/мс]

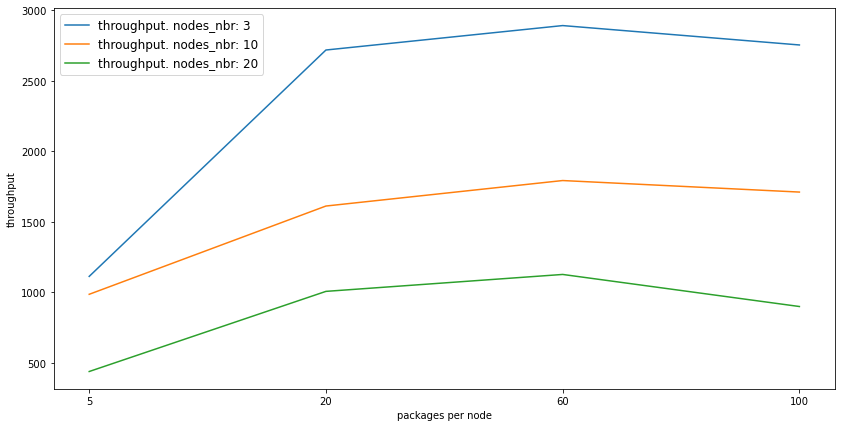
AverageLatency в размерности [мс]

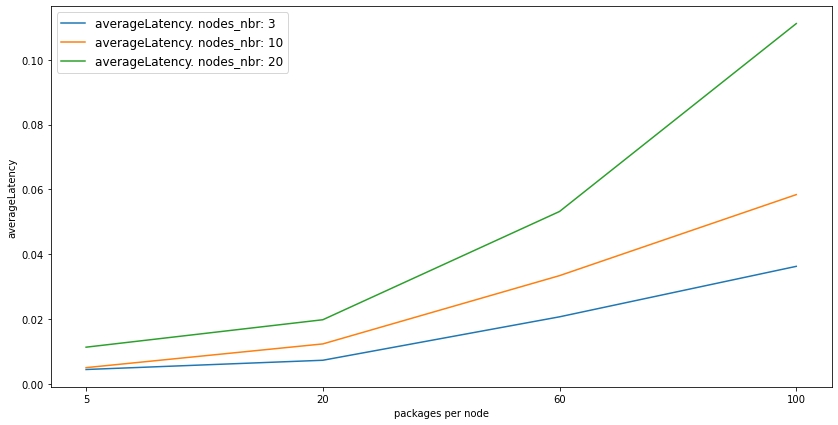
****

****

Из графиков видно, что при росте числа узлов пропускная способность падает, а задержка растет.

При изучении логов было замечено, что основное время обработки затрачивается на чтение из очереди, сбор статистики и перемещение пакета в следующую очередь. Из-за этого все сообщения скапливаются в очереди на одном узле, потом перемещаются на другой узел и так далее. Поэтому при равном количестве пакетов система с большим числом узлов показывает худшие характеристики.

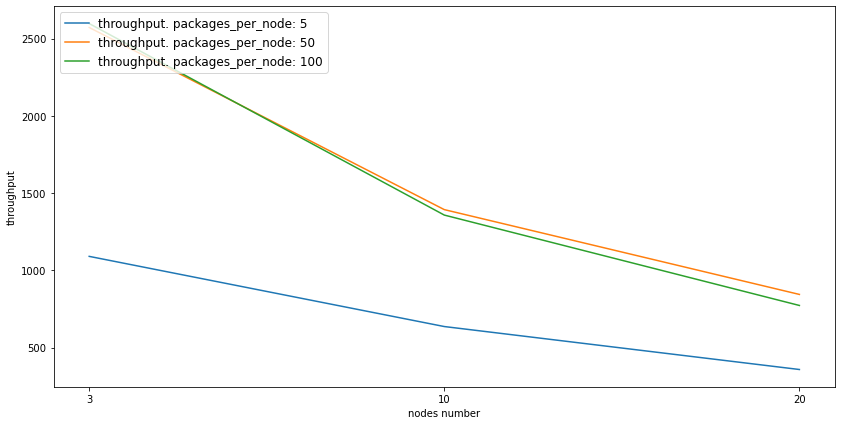
****

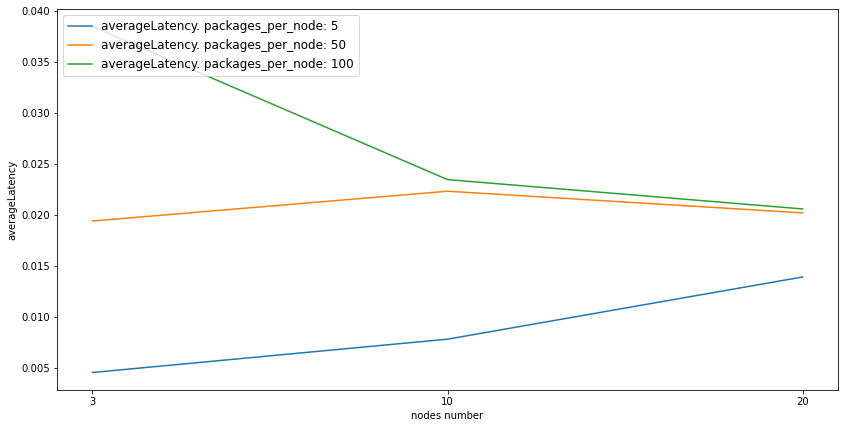
****

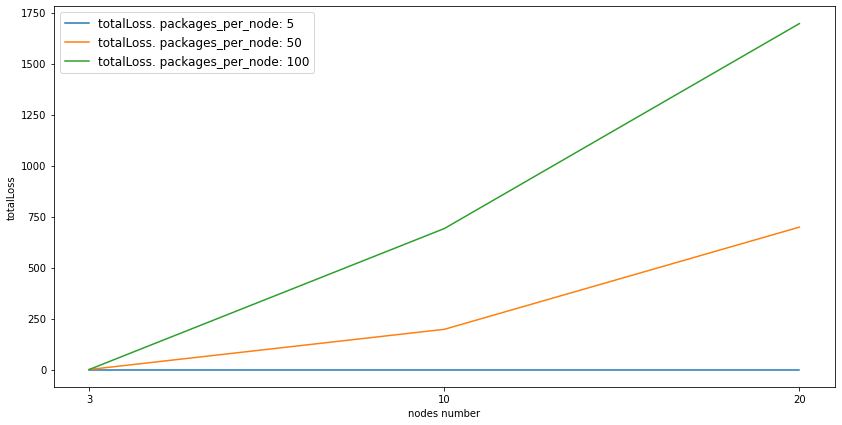
При фиксированном числе узлов, при росте числа пакетов задержка увеличивается, а пропускная способность сначала увеличивается, а потом выходит на плато и начинает убывать. Это происходит из-за того, что сначала система разрежена, но по мере роста числа пакетов, через один узел проходит большее количество узлов, что увеличивает пропускную способность.

Также можно отметить, что при выходе throughput на плато latency начинает линейно расти с увеличением числа пакетов в системе.

**Графики и статистика (ArrayBlockingQueue)**



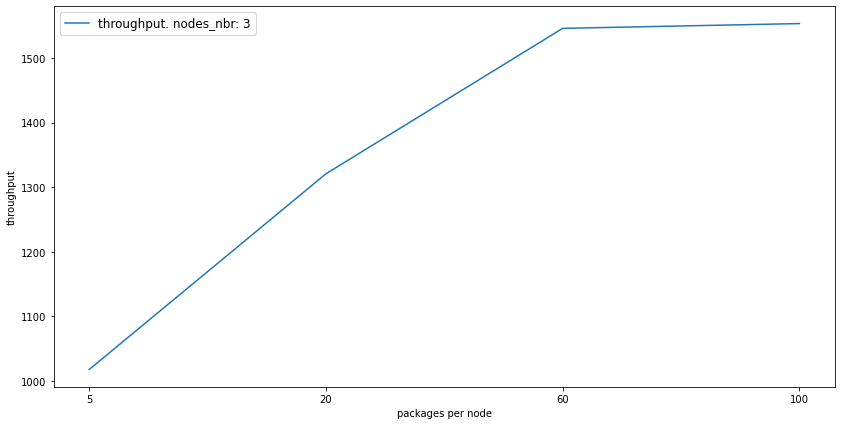


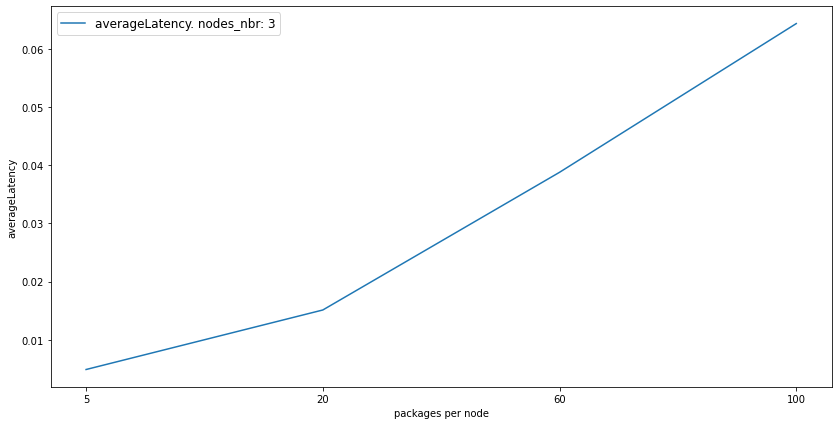


Прежде всего стоит обратить внимание на последний график. На нем видно, что если в системе число пакетов больше размера входной очереди узла, пакеты начинают теряться. Поэтому можно полностью доверять только графику для 3 узлов доверять.

По графику для 3х узлов видно, что поведение arrayBlockingQueue схоже с поведением для linkedBlockingQueue. А из-за потери пакетов при большем количестве узлов наблюдается переломы в графике времени задержки.

Подробный график зависимости throughput и latency от числа пакетов при 3х узлах.

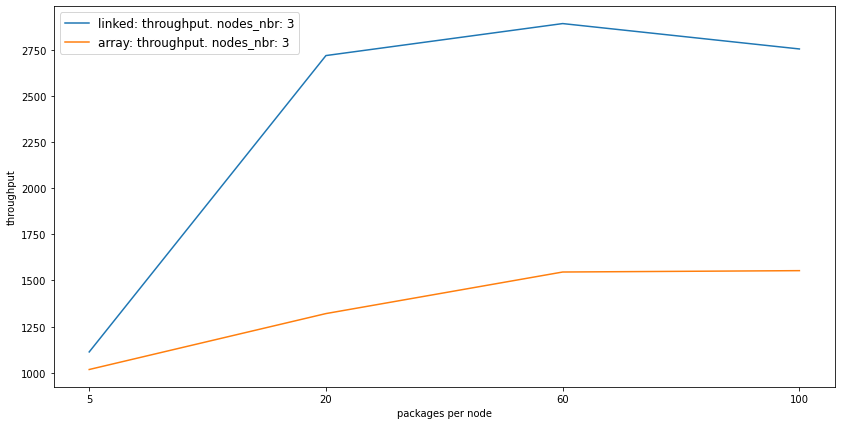


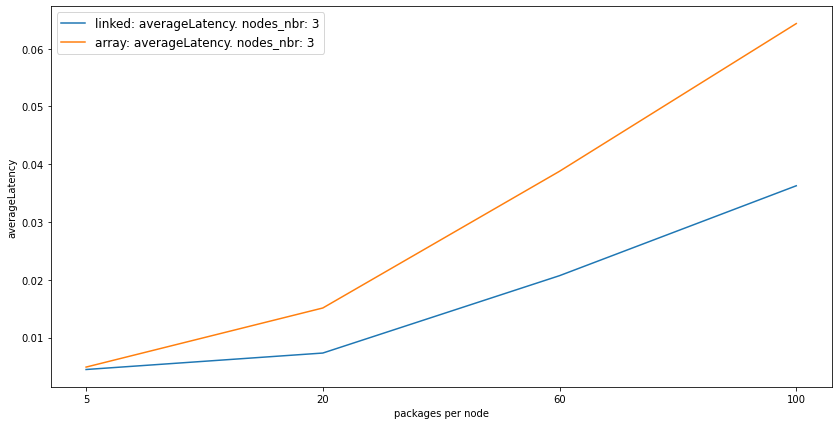


На графиках видно выход throughput на плато и линейность latency.

**Сравнение реализаций**

Сравнивая характеристики LinkedBlockingQueue и ArrayBlockingQueue, можно заметить, что показатели LinkedBlockingQueue лучше. При одинаковых параметрах системы throughput выше, а latency ниже. Это сходится с теорией, так как ArrayBlockingQueue построен на одном локе с двойным условием, а LinkedBlockingQueue на двух локах – один для записи, второй для чтения. За счет этого throughput LinkedBlockingQueue выше.





**Параметр оптимизации.**

Для повышения throughput и снижения latency при загруженности системы я ввел параметр **process\_per\_time.** Этот параметр отвечает за количество пакетов, вычитываемых узлом из очереди до того, как выйти из функции process() и проверить флаг AtomicBoolean остановки процесса.

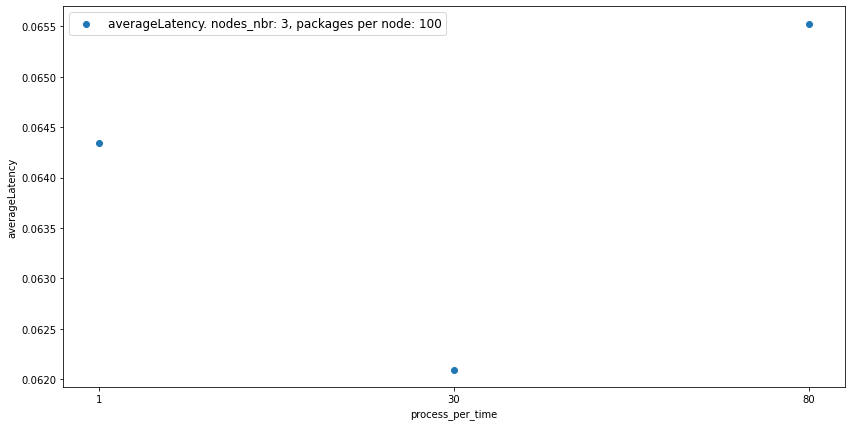
private void process() {  
 DataPackage dataPackage;  
 try {  
 dataPackage = bufferStack.poll(*POLL\_TIMEOUT\_MS*, TimeUnit.*MILLISECONDS*);  
 } catch (InterruptedException ex) {  
 *log*.error("{} Caught interrupted exception during taking from blocking queue", logPrefix, ex);  
 return;  
 }  
 if (dataPackage == null) {  
 return;  
 }  
 sendToNextNode(dataPackage);  
 for (int i = 1; i < processingPerTime; i++) {  
 dataPackage = bufferStack.poll();  
 if (dataPackage == null) {  
 return;  
 }  
 sendToNextNode(dataPackage);  
 }  
}

Я проводил измерение статистики для системы из 3 узлов и 300 пакетов, изначально распределенных поровну между узлами. Статистика собиралась для **process\_per\_time = 1, 30, 80.**

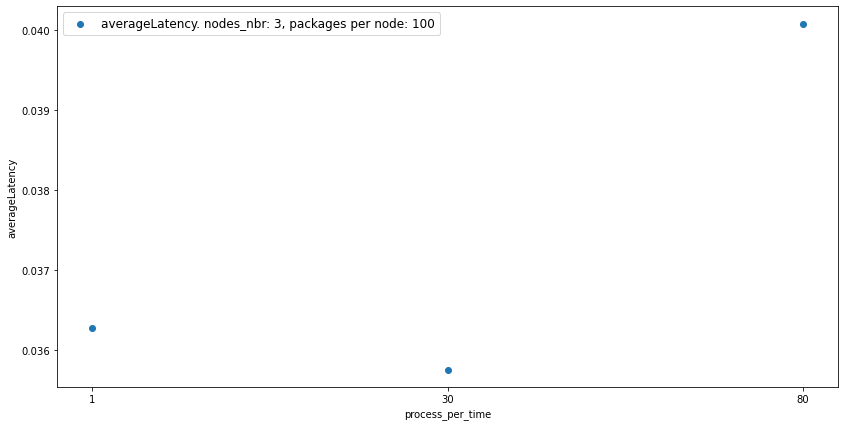
В результате получились следующие графики:

1. Для ArrayBlockingQueue





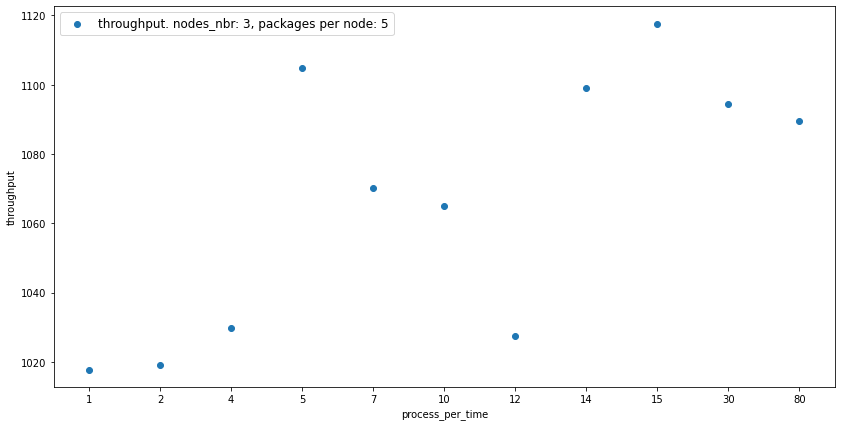
1. Для LinkedBlockingQueue:





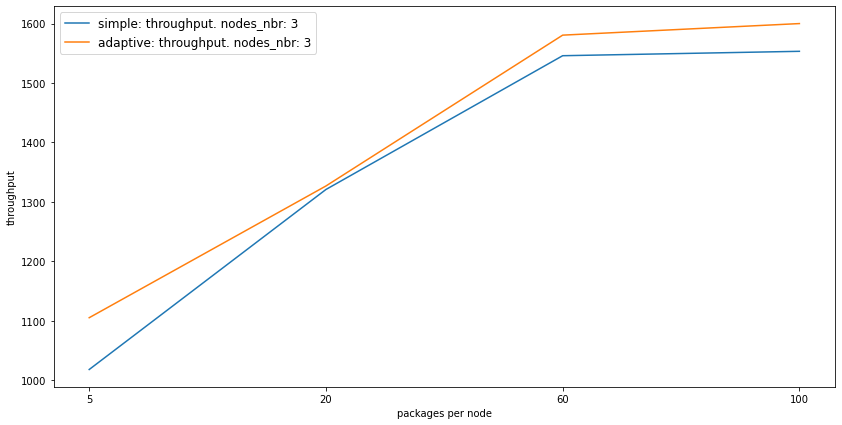
Видно, что при увилечении process\_per\_time характеристики улучшаются, но при слишком большом значении становятся еще хуже чем при process\_per\_time = 1.

Изучим более подробно график для системы из 3 узлов и 15 пакетов:



Видно, что пик throughput приходится на process\_per\_time, равный сумме пакетов в системе, а так же на число пакетов, приходящихся на 1 узел.

Ниже приведены графики для process\_per\_time равного сумме пакетов



**Вывод:**

В работе была реализована модель token-ring, приведены 2 способа реализации: с помощью ArrayBlockingQueue и LinkedBlockingQueqe.

2й вариант показал себя лучше по всем показателям. Также в ходе исследования было замечено что с ростом числа узлов в системе  throughput  падает, а latency растет, а при росте числа пакетов в системе throughput растет, пока не выйдет на свой максимум, а latency увеличивается.

В системе был задействован параметр process\_per\_time, позволяющий проводить оптимизацию, основываясь на текущей загруженности и количестве пакетов в ситеме.

Оптимизация оказала значительное влияние только на реализацию с ArrayBlockingQueue.

Работу выполнил:

Бочаров Георгий, группа М05-114д