АНАЛИЗ ПАРАМЕТРОВ СИСТЕМ ДЕТЕКТИРОВАНИЯ МНОЖЕСТВЕННЫХ ВИЗУАЛЬНЫХ ОБЪЕКТОВ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

В.И. Проценко[1], Н.Л. Казанский[1,2], П.Г. Серафимович[1,2]

[1] Самарский государственный аэрокосмический университет имени академика С.П. Королёва (национальный исследовательский университет) (СГАУ), Самара, Россия,

[2] Институт систем обработки изображений РАН, Самара, Россия

Аннотация

Проведён анализ параметров систем детектирования множественных объектов на основе фреймворков для потокового анализа данных Apache Storm и IBM InfoSphere Streams. В качестве объектов для детектирования были выбраны изображения лиц. Тестирование производилось на кластере, состоящем из пяти 32-ядерных узлов под управлением операционной системы CentOS. Apache Storm показал лучшую масштабируемость по сравнению с IBM InfoSphere Streams. Для изображения размером 1920×1080 достигнута скорость обработки 24 кадра в секунду при использовании Apache Storm.

Ключевые слова: большие данные, потоковая обработка, набор изображений, детектирование лиц, пропускная способность, система реального времени.

Цитирование: **Проценко, В.И.** Анализ параметров систем детектирования множественных визуальных объектов в режиме реального времени / В.И. Проценко, Н.Л. Казанский, П.Г. Серафимович // Компьютерная оптика. – 2015. – Т. 39, № 4. – С. 582-591. – DOI: 10.18287/0134-2452-2015-39-4-582-591.

Введение

На сегодняшний день основной объём информации, появляющейся в сети интернет, генерируется пользователями с мобильных устройств. В основном это изображения, звук и видеоданные. Согласно прогнозам компании Cisco, величина мобильного мультимедиа трафика к 2019 году испытает экспоненциальный рост, достигнув значений 23,4 экзабайт в месяц [1]. Важность актуальной информации в сочетании с высокой интенсивностью появления данных создают проблему анализа информации в реальном времени. Для решения данной проблемы был создан ряд систем потоковой обработки данных как коммерческих, так и с открытым кодом. Среди наиболее известных систем с открытым кодом можно назвать Apache Storm, Apache Samza, S4, Apache Streaming, а среди коммерческих – IBM InfoSphere Streams, Oracle Event Processing, Microsoft StreamInsight, Amazon Kinesis, MillWheel.

В настоящей статье выполнен анализ параметров систем детектирования множественных объектов в последовательности изображений на основе фреймворков для потокового анализа: Apache Storm и IBM InfoSphere Streams. Apache Storm является одним из наиболее популярных продуктов с открытым кодом, а IBM InfoSphere Streams – среди коммерческих. Произведён анализ характеристик систем на четырёх размерах изображений: на небольшом размере 100×100, на размере 640×640, используемом в социальной сети Instagram, на размере 1920×1080, известном как Full HD, и на размере 4096×3112, являющимся полнокадровым разрешением 4K. Сравнение произведено на последних на текущий момент версиях: 0.9.4 для Apache Storm и v4.0 IBM InfoSphere Streams.

Рассматриваемые фреймворки анализировались авторами ранее в статье [2], результаты которой показали меньшую требовательность к объёму оперативной памяти для IBM InfoSphere Streams по сравнению с Apache Storm, примерно в 3-4 раза. При этом Apache Storm был более производительным. Однако из-за проведения эксперимента на виртуальной машине исследовать пропускную способность в зависимости от параллелизма не представлялось возможным. Данная работа дополняет ранее выполненный анализ систем новыми результатами экспериментов. Также в описании архитектур приводится информация, соответствующая последним версиям фреймворков.

Анализ визуальной информации имеет важное значение и широкий спектр применений [3–7]. Детектирование визуальных объектов находит применение в таких задачах, как видеомониторинг общественных и частных пространств [8–11], мониторинг дорожного трафика [12–16], улучшение взаимодействия с пользователем [17–18], в биометрических системах [19–20], в системах контроля качества [21–24], автоматического управления транспортным средством [25–26]. Для детектирования объектов в данной работе используется реализация каскадного классификатора библиотеки OpenCV [27], алгоритм которого впервые был описан в работе P. Viola и M. Jones [28], а затем улучшен в работе R. Lienhart [29]. Для данного классификатора объект, который необходимо найти на изображении, описывается конфигурационным файлом – решающим деревом особенностей. Дерево особенностей можно получить после фазы обучения на обучающей выборке. В данной работе был взят готовый конфигурационный файл для детектирования лица в анфас за авторством Rainer Lienhart [30].

При работе с задачами обработки видеопотоков в большинстве случаев возникает требование обработки данных в реальном времени. Для выполнения данного требования применяются различные подходы: создание распределённых алгоритмов на основе технологии MPI [31–32], использование технологий CUDA и OpenCL для графических процессоров [33–34], ис-

пользование программируемых логических интегральных схем типа FPGA [35 – 36], использование встроенных сопроцессоров [12]. Недостатком MPI-подхода является сложность создания алгоритмов, эффективно масштабируемых в кластерах с гетерогенной структурой, а также алгоритмов, устойчивых к возможным сбоям узлов. Применение специализированных процессоров позволяет добиться значительной скорости обработки, от 30 до 160 кадров в секунду, в задаче слежения за движущимися объектами [12 – 14], однако такой подход ускоряет обработку только на одном узле и не решает проблему распределения вычислений по множеству компьютеров, усложняет реализацию обработки, учитывающую другие типы информации, например текст, GPS-координаты.

Далее будут описаны архитектуры двух рассматриваемых фреймворков – Apache Storm и IBM Info–Sphere Streams, произведётся сравнение систем детектирования множественных объектов по параметрам задержки на обработку кадра и пропускной способности.

Архитектуры Apache Storm и IBM InfoSphere Streams

Apache Storm

Так как архитектура уже была описана авторами ранее в статье [2], мы приведём только отличия, касающиеся версии 0.9.4, и описание механизма надёжной обработки кортежей, используемого в эксперименте. Отличием является то, что для передачи данных между процессами механизм передачи сообщений Netty [37] теперь используется по умолчанию. Также в новой версии исправлено множество ошибок. Отметим положительное свойство системы – устойчивость к выходу из строя узлов и каналов передачи данных. Благодаря тому, что процессы Nimbus и Supervisor хранят состояние не локально, а в Zookeeper [38], система может восстанавливать работоспособность после аварийного завершения процессов.

В данной работе в эксперименте будет использоваться механизм подтверждений. При использовании механизма подтверждений система позволяет добиться гарантии обработки кортежа один или более раз (at-least once).

Для того, чтобы описать алгоритм подтверждения обработки, введём соответствующую терминологию. Id_i – идентификатор, присваиваемый пользователем кортежу в источнике Spout. X_i будет обозначать случайное число, идентификатор «дерева» зависимостей, Y_j – случайное число, идентификатор кортежа j, K_j – кортеж j, где i и j – натуральные числа. Рассмотрим топологию с одним источником source, двумя обработчиками Bolt1, Bolt2 и обработчиком Acker, необходимым для работы алгоритма подтверждений.

На рис. 1 проиллюстрирована работа топологии в режиме гарантии обработки. В работе алгоритма можно выделить 3 фазы: начало обработки, связывание кортежей зависимостью и подтверждение обработки. В начале для нового кортежа источника K_1 и идентификатора Id_1 генерируется пара $<X_1, Y_1>$, в источнике сохраняется соответствие Id_1 и X_1, а пара $<X_1, Y_1>$ передаётся обработчику Acker. Далее кортеж K_1 передаётся с парой $<X_1, Y_1>$ всем последующим обработчикам. Для кортежа K_2 в Bolt1 генерируется Y_2 и пара $<X_1, Y_2>$ передаётся Acker. Затем обработчик Bolt1 передаёт Bolt2 кортеж K_2 с парой $<X_1, Y_2>$ и подтверждает обработку K_1, передав пару $<X_1, Y_1>$ в обработчик Acker. На заключительном этапе обработки Bolt2 подтверждает обработку K_2 аналогичным способом.

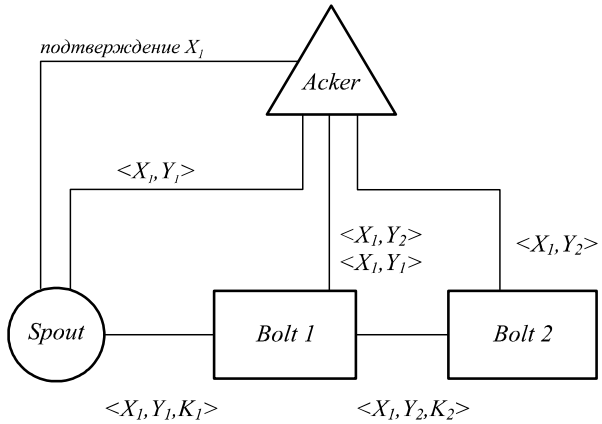

Рис. 1. Иллюстрация работы механизма подтверждений Apache Storm

Опишем недостающую логику алгоритма – поведение Acker и Spout при получении сообщения. Acker хранит пары в таблице. В таблицу добавляется пара $<X_i, Y_j>$ из сообщения, если ключ X_i отсутствует в ней. Иначе к значению Y_k из существующей записи $<X_i, Y_k>$ и значению Y_j из сообщения применяется операция «исключающее или» (^) и результат сохраняется под тем же ключом. В рассматриваемом примере после получения второй пары в таблице хранится запись $<X_1, Y_1 \wedge Y_2>$. После получения подтверждения от Bolt1 – хранится $<X_1, Y_2>$ и после получения от Bolt2 – хранится $<X_1, 0>$. Acker посылает источнику сообщение об успешном завершении X_i, если значение по ключу стало равным нулю, либо о неудаче, если запись находилась в таблице больше фиксированного времени. После получения сообщения от Acker источник вызывает пользовательскую функцию ack, либо fail с параметром Id_i и удаляет связь между Id_i и X_i.

Использование данного механизма также позволяет контролировать количество обрабатываемых кортежей в системе, то есть тех, которые отправлены из источника, но ещё не подтверждены (параметр topology.max.spout.pending). Это позволяет контролировать размер используемой оперативной памяти источника и избежать ошибок OutOfMemoryException при интенсивности источника большей, чем производительность обрабатывающего подграфа.

IBM InfoSphere Streams

В новой версии v4.0 было введено понятие домена. Домен объединяет ресурсы кластера и сервисы домена. Instance объединяет подмножество ресурсов домена и процессы, отвечающие за потоковую обра-

ботку. Одному домену может принадлежать несколько Instance. На рис. 2 изображена архитектура системы. Часть сервисов, ранее принадлежащих Instance, сделаны процессами домена: SWS – веб-консоль администратора, AAS – сервис авторизации пользователей, JMX – jmx-расширения для управления сервисами, AUDITLOG – сервис, ведущий лог операций, производимых над компонентами фреймворка. Конфигурационная информация и данные о назначенных заданиях хранятся в Zookeeper.

В Instance на управляющем узле запускается четыре процесса: сервис метрик приложения, менеджер приложения, планировщик и сервис представления. Менеджер приложения, SAM, отвечает за назначение, перераспределение и отмену задач пользовательского приложения. Он взаимодействует с планировщиком для расчёта распределения задач и с контроллерами рабочих узлов для запуска или удаления задач. Сервис метрик приложения, SRM, следит за работой всех остальных сервисов, собирает данные об их выполнении, а также собирает данные о производительности пользовательских задач, взаимодействуя для этого с контроллерами рабочих узлов HC. Планировщик SCH, анализируя данные о ресурсах системы от сервиса метрик приложения, рассчитывает распределение задач для менеджера приложения. Сервис представления VIEW управляет данными, используемыми в представлениях, которые, в свою очередь, описывают набор атрибутов для отображения на графиках, в таблицах и в Microsoft Excel.

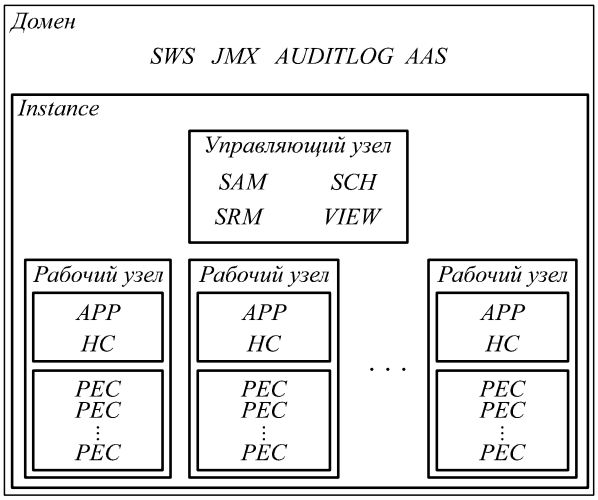

Рис. 2. Архитектура InfoSphere Streams. Домену принадлежит Instance. Верхним прямоугольником в Instance изображается узел с управляющими сервисами, нижними прямоугольниками изображены исполнительные узлы, каждый с локальным управляющим процессом (Host Controller) и рабочими процессами PEC

На рабочих узлах запускаются контроллеры HC, в ответственность которых входят запуск, остановка и мониторинг задач. В его обязанности также входит сбор метрик задач и передача их сервису метрик на управляющий узел. Кроме контроллера, на рабочих узлах запускается сервис развёртывания приложения APP. Данный сервис подготавливает ресурсы для развёртывания и запуска задач, размещает скомпилированные блоки кода к соответствующим ресурсам.

Пользовательская программа, также как и в Apache Storm, представляет из себя граф, описанный на языке SPL [39], в узлах которого находятся операторы. Операторы могут быть реализованы на языках C++ и Java. Для реализации на других языках фреймворк специальных средств не предоставляет, они могут выполняться в системе обёрнутыми в Java или C++. Результатом компиляции является набор разделяемых библиотек (PE). Задачи, упоминаемые выше, и processing element PE в данной секции являются синонимами. Задачи выполняются в процессах PEC.

Начиная с версии 4.0 в IBM InfoSphere Streams реализован механизм надёжной обработки кортежей, позволяющий добиться гарантий обработки один или более раз (at-least-once) [40]. Операторы, начиная с которых вся последующая обработка должна быть надёжной, помечаются ключевым словом @consistent. Начиная с нужного оператора также можно отменить работу логики подтверждений, пометив его ключом @autonomous. Кроме того в каждом приложении с аннотацией @consistent должен фигурировать оператор JobControlPlane. Множество операторов, заключённое между @consistent и последним или @autonomous оператором, называется consistent region и гарантирует обработку каждого кортежа хотя бы один раз в условиях возможных сбоев узлов или каналов передачи сообщений. Реализация алгоритма основана на работе Chandy и Leslie Lamport [41].

Описание модельной задачи

Эксперимент ставился для задачи подсчёта количества лиц в потоке изображений. Лица детектировались каскадным классификатором из библиотеки OpenCV. На вход данному классификатору подавался файл конфигурации, натренированный для детектирования лица анфас. Результатом работы классификатора является список прямоугольников (координаты противоположных по диагонали углов), количество которых считалось количеством лиц на изображении. На рис. 3 показан результат работы алгоритма детектирования лиц.

В табл. 1 приведено время детектирования на одном узле для рассматриваемых разрешений.

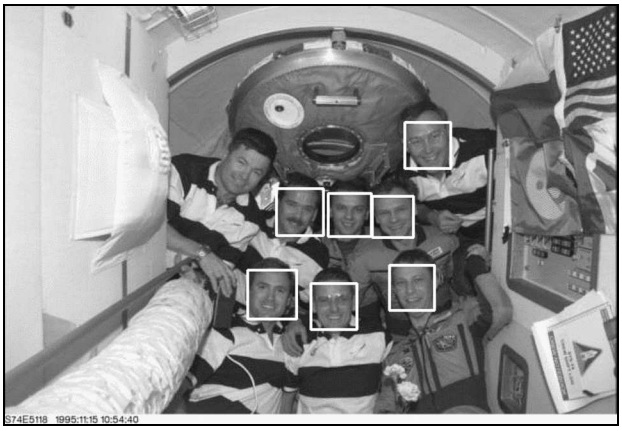

Рис. 3. Результат детектирования лиц на изображении

Табл. 1. Среднее время детектирования лиц на одном ядре для разных размеров изображения

Размер изображения	Время обработки (сек)
100×100	0,008
640×640	0,417
1920×1080	1,621
4096×3112	9,988

Вычислительный эксперимент

Описание

Единицей измерения пропускной способности было *количество обработанных изображений в секунду*. Величина задержек измерялась в секундах. Задержка рассчитывалась как разница между моментом времени отправки кортежа из источника и отправки кортежа из обработчика.

Топология Apache Storm была реализована на языке Java, состояла из одного источника Spout и одного обработчика Bolt с указанием степени параллелизма. Аналогичная программа была написана для InfoSphere Streams. Операторы были реализованы на языке Java, связи между операторами описаны на языке SPL. В обоих алгоритмах передача производилась в сериализованном формате: ширина, высота, целое число – тип OpenCV, массив байтов изображения.

В ходе планирования и проведения экспериментов было выявлено несколько ключевых параметров, влияющих на анализируемые показатели. В обеих системах существует возможность написания программы с использованием механизма гарантии обработки и без него. Эксперимент для системы Apache Storm без гарантии обработки «один или более раз» не проводился, так как источник в данной системе работает асинхронно с механизмом передачи сообщений, что приводило к заполнению всей выделенной памяти непереданными кортежами и завершению работы приложения с ошибкой OutOfMemoryException. В отличие от режима с гарантией обработки, в котором контроль за одновременно обрабатываемым количеством кортежей есть, в этом режиме его пришлось бы писать самому, что усложнило бы эксперимент. В случае с системой IBM Streams использование @consistent приводило к уменьшению пропускной способности в 1,1 – 1,2 раза.

Важным параметром также был параметр topology.max.spout.pending для Apache Storm. Было выявлено, что при установке данного значения равным topology.max.spout.pending=2parallelism достигалась высокая пропускная способность, и в то же время задержка сохранялась небольшой относительно степени параллелизма. Дальнейшее увеличение приводило к увеличению задержки и незначительному приросту пропускной способности, которое для topology.max.spout.pending = 3parallelism стабилизировалось около определённого значения. Кроме того, от увеличения данного параметра линейно зависит потребление оперативной памяти, выделенной для JVM источника Spout. Отметим также, что установка значения меньшего, чем значение степени параллелизма, приведёт к параллельной обработке входного потока только частью обработчиков единовременно.

Таким образом, далее анализируются результаты выполнения программы с гарантией обработки кортежей с удвоенным значением одновременно обрабатываемых кортежей, по сравнению с степенью параллелизма для Apache Storm и программы IBM InfoSphere Streams без наличия операторов с гарантией обработки сообщений.

Для анализа систем было добавлено журналирование в каждом операторе: источник для каждого кортежа записывал время начала отправки, обработчики записывали текущее время после отправки результата распознавания объекта (количество лиц на изображении). После минутной работы алгоритм останавливался, и результаты с каждого узла объединялись в единый файл для последующего анализа. Важным моментом при данном способе замера времени является синхронизация времени на узлах с достаточной для эксперимента точностью, что было проделано перед проведением замеров. Узлы были синхронизированы посредством NTP с точностью до третьего знака после запятой.

Тестирование проводилось на кластере с пятью узлами под управлением операционной системы CentOS. Расчёты проводились на 4 узлах, оставляя пятый в качестве управляющего. Характеристики кластера: скорость сети – 10 Гбит/с, процессоры на каждом узле – 2 процессора Intel Xeon E5-2450v2 по 8 реальных ядер с включённой технологией HyperThreading, размер оперативной памяти каждого узла – 96 Гб. Использовалась версия 2.4.10 библиотеки OpenCV.

Анализировались задержки обработки изображения и пропускная способность систем в зависимости от количества параллельно работающих обработчиков на четырёх изображениях разного разрешения. В силу того, что мы хотели бы дать гарантии максимальной задержки, для задержек анализировались значения 95-го процентиля, которое мы будем обозначать греческой буквой α. Данное значение обозначает промежуток времени, в течение которого порядок обрабатываемых кортежей не гарантируется. То есть кортеж K2, отправленный из источника через промежуток времени, отсчитываемый от K1 и равный 95-му процентилю задержки кортежа, с вероятностью 0,95 будет обработан позже K1. Это является ориентиром для выбора буфера, необходимого для восстановления порядка кортежей, при обработке видеопотоков в реальном времени.

Результаты

Сначала рассмотрим графики пропускной способности двух систем, которые представлены на рис. 4*а – г*. Основное, что бросается в глаза, – это ограничение роста пропускной способности программы для IBM InfoSphere Streams. Узким местом здесь является источник. Со значения параллелизма 16 нить источника начинает сильно загружать ядро процессора, и к 32 загрузка составляет больше чем 90 %.

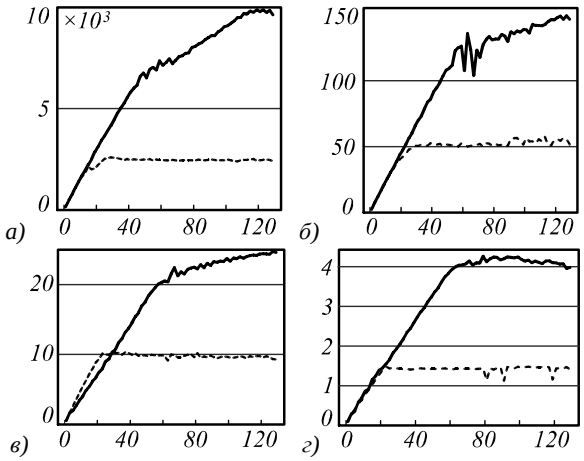

Рис. 4. Зависимость пропускной способности от степени параллелизма для разных размеров изображений: а) 100×100, б) 640×640, в) 1920×1080, г) 4096×3112. Сплошной линии соответствует Apache Storm, пунктирной – InfoSphere Streams. По оси абсцисс изменяется степень параллелизма, по оси ординат – пропускная способность (кадр/с)

На рис. 5 и в табл. 2 представлен вывод программы *top* о загрузке процессора для узла с источником, на котором видно, что только половина обработчиков на данном узле загружена. Значения загрузки более 100 % обозначают загрузку более чем одного ядра. Так как новые изображения поступают с меньшей скоростью, чем производится обработка, часть обработчиков простаивает.

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13737 streamsa  20   0 5356m 412m  42m R 100.4  0.4   1:27.51 Thread-12
13735 streamsa  20   0 5356m 415m  42m R 100.1  0.4   1:29.06 Thread-12
13733 streamsa  20   0 5356m 413m  42m R 100.1  0.4   1:29.79 Thread-12
13740 streamsa  20   0 5544m 290m  42m R 92.8  0.3   2:10.04 ImageSource-thr
13739 streamsa  20   0 5356m 415m  42m R 71.4  0.4   1:24.53 Thread-12
13731 streamsa  20   0 5210m 255m  42m S 52.0  0.3   1:32.60 Thread-12
13729 streamsa  20   0 5210m 252m  42m S 14.2  0.3   1:32.36 Thread-12
14236 root      20   0 16360 2652 1004 R  1.3  0.0   0:00.39 top
```

Рис. 5. Загруженность процесса источника IBM InfoSphere Streams. Нити отсортированы по убыванию загрузки процессора. Источник имеет название ImageSource-thr, а обработчики – Thread-12

Табл. 2. Данные о загруженности процессора из утилиты top для узла с источником IBM InfoSphere Streams. Нити отсортированы по убыванию загрузки процессора. Источник имеет название ImageSource-thr, а обработчики – Thread-12

PID	Загруженность процессора (%)	Имя команды
13737	100,4	Thread-12
13735	100,1	Thread-12
13733	100,1	Thread-12
13740	92,8	ImageSource-thr
13739	71,4	Thread-12
13731	52,0	Thread-12
13729	14,2	Thread-12

В отсутствие обработчиков и, следовательно, необходимости передавать сообщения на размере 640×640 производительность источника была равна 365 кадров/сек. Это говорит о том, что проблемным местом здесь является механизм передачи сообщений IBM Streams. К сожалению, параметров для тонкой настройки данного механизма IBM InfoSphere Streams не предоставляет. Поэтому для данного эксперимента – топологии с одним источником и множеством однотипных обработчиков – обойти данное ограничение не получилось. Это поведение было характерно для всех размеров изображений.

Графики программы для Apache Storm можно охарактеризовать двумя прямыми с разными углами наклона, сменяющими друг друга на отметке значения параллелизма 64. Данное явление объясняется включенной технологией HyperThreading. До значения 64, равного количеству ядер кластера, каждый поток рассчитывается на своём ядре, но после этого значения потоки начинают делить физические ядра [42]. Технология HyperThreading позволяет добиться ускорения за счёт объединения потоков инструкций нескольких нитей в один для того, чтобы загрузить все специализированные вычислительные блоки ядра. Можно предположить, что наблюдаемое уменьшение наклона прямой, характеризующей пропускную способность после параллелизма 64, при увеличении размера изображений связано с увеличением времени однотипных операций и в итоге уменьшению ускорения, даваемого технологией HyperThreading из-за однородности потоков инструкций. В табл. 3, 4, 5, 6 представлена выборочная часть результатов замеров пропускной способности для значений параллелизма от 1 до 128, идущих с шагом 20. Выбор данного набора значений параллелизма для таблиц обусловлен желанием компактного представления данных, дополняющих информацию, представленную на графиках.

Табл. 3. Пропускная способность для размера изображения 100×100

Степень параллелизма	Пропускная способность	
	Storm	IBM Streams
1	162	168
21	3112	2222
41	5799	2455
61	7140	2462
81	8072	2444
101	9084	2491
121	9875	2458

Табл. 4. Пропускная способность для размера изображения 640×640

Степень параллелизма	Пропускная способность	
	Storm	IBM Streams
1	2,5	2,5
21	47,1	42,2
41	91,0	51,6
61	107,7	52,1
81	131,7	52,0
101	138,8	56,6
121	147,9	55,8

На рис. 6*а – г* изображены графики 95-го процентиля для задержек на обработку изображения. В табл. 7 – 10 представлена выборочная часть результатов 95-го процентиля для задержек. На изображениях 100×100 значения α для обеих систем были примерно одинаковыми и отличались только на небольших значениях

параллелизма. На всех графиках в начале программа для IBM Streams имела большее значение α, а затем α стабилизировалось около определённой величины. В силу того, что существовало ограничение на производительность источника в IBM Streams, нельзя сказать, что в данной задаче задержки более предсказуемые по сравнению с Apache Storm.

Табл. 5. Пропускная способность для размера изображения 1920×1080

Степень параллелизма	Пропускная способность	
	Storm	IBM Streams
1	0,56	0,55
21	7,46	9,34
41	14,48	9,95
61	20,51	9,80
81	22,23	10,18
101	23,33	9,56
121	24,40	9,57

Табл. 6. Пропускная способность для размера изображения 4096×3112

Степень параллелизма	Пропускная способность	
	Storm	IBM Streams
1	0,09	0,09
21	1,44	1,39
41	2,72	1,43
61	3,93	1,40
81	4,12	1,13
101	4,18	1,45
121	4,09	1,42

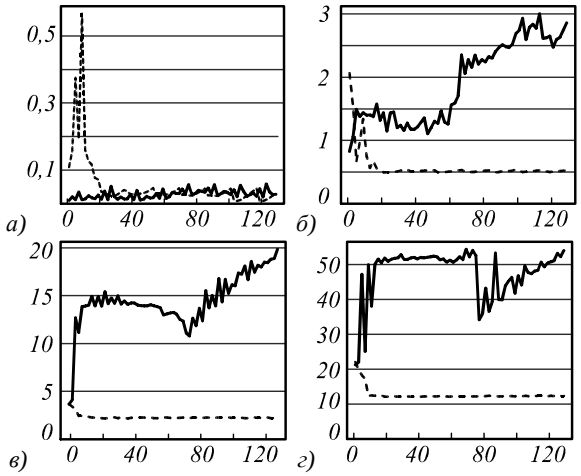

Рис. 6. 95-й процентиль для задержки на обработку изображений для двух систем в зависимости от степени параллелизма для разных размеров изображений: а) 100×100, б) 640×640, в) 1920×1080, г) 4096×3112. Сплошной линии соответствует Apache Storm, пунктирной – InfoSphere Streams. По оси абсцисс изменяется степень параллелизма, по оси ординат – 95-й процентиль для задержки (с)

Начиная с размера 640×640 на графиках для программы под Apache Storm видно изменение характера задержек после параллелизма 64. Во всех случаях α задержек начинает расти, что соответствует тому, что нити начинают делить физические ядра. Интересным эффектом является то, что для 1920×1080 и 4096×3112 значение α находится на определённом уровне, затем испытывает спад, и уже после этого медленно растёт. Если бы мы привели среднюю задержку, мы бы увидели, что средняя задержка держится на определённом уровне, а затем после значения параллелизма 64 испытывает медленный рост. Можно сделать вывод, что количество долгих задержек для крупных изображений больше, что влияет на показатель, так как объём обработанных изображений меньше.

Табл. 7. 95-й процентиль для задержек для двух систем на размерах изображений 100×100

Степень параллелизма	95-й процентиль задержек (сек)	
	Storm	IBM Streams
1	0,010	0,109
21	0,013	0,036
41	0,012	0,029
61	0,016	0,027
81	0,040	0,033
101	0,027	0,045
121	0,020	0,008

Табл. 8. 95-й процентиль для задержек для двух систем на размерах изображений 640×640

Степень параллелизма	95-й процентиль задержек (сек)	
	Storm	IBM Streams
1	0,827	2,063
21	1,440	0,497
41	1,212	0,515
61	1,563	0,509
81	2,242	0,520
101	2,742	0,525
121	2,474	0,507

Табл. 9. 95-й процентиль для задержек для двух систем на размерах изображений 1920×1080

Степень параллелизма	95-й процентиль задержек (сек)	
	Storm	IBM Streams
1	3,667	3,680
21	13,894	2,213
41	14,248	2,256
61	13,088	2,281
81	13,687	2,270
101	16,139	2,248
121	18,496	2,264

Табл. 10. 95-й процентиль для задержек для двух систем на размерах изображений 4096×3112

Степень параллелизма	95-й процентиль задержек (сек)	
	Storm	IBM Streams
1	21,265	22,051
21	50,92	12,127
41	52,037	12,130
61	50,647	12,217
81	43,251	12,294
101	46,809	12,352
121	50,766	12,422

Выводы

Выполнен анализ систем детектирования множественных визуальных объектов, построенных на основе фреймворков Apache Storm и IBM InfoSphere Streams. В ходе вычислительного эксперимента подсчитывалось количество лиц в потоке изображений. В работе анализировались параметры пропускной способности систем, а также 95-й процентиль для задержек на обработку изображения. Новизной данной работы является использование систем потокового анализа, которые лишены недостатков других используемых для этой задачи технологий. В частности, сложности организации передач данных при использовании MPI. А также проблемы длительной задержки при блочной обработке на Hadoop. Разработанные системы позволяют детектировать объекты в видеопотоках в реальном времени.

В ходе экспериментов было выявлено, что система IBM InfoSphere Streams для данного эксперимента не масштабируется на число ядер в кластере большее, чем 16. На больших изображениях преимуществ от использования технологий HyperThreading нет из-за однородности потока инструкций. Система Apache Storm показала линейный рост пропускной способности до степени параллелизма 64 для всех размеров изображений. При использовании Apache Storm для изображения размером 1920×1080 достигнута скорость обработки 24 кадра в секунду. Данная производительность позволяет обрабатывать видеопоток с разрешением FullHD в реальном времени. Таким образом, для рассматриваемой задачи лучше использовать систему Apache Storm.

Если сравнивать с предыдущими подходами [31 – 36] к обработке данных в реальном времени, то преимуществом обработки в анализируемых потоковых фреймворках является возможность совместного анализа потоков разного типа, гарантии обработки данных, отсутствие единой точки отказа, возможность работы с данными, не помещающимися в оперативную память одного узла. Фреймворки могут быть использованы для сложной обработки изображений и видеопотоков, включая учёт информации других типов, например GPS-координат, звука и текста.

Дальнейшие исследования могут быть проведены в направлении усложнения логики обработки изображений, использования графических процессоров на обрабатывающих узлах, что может увеличить производительность систем в несколько раз [34, 43].

Благодарности

Исследование выполнено при поддержке Российского научного фонда (проект № 14-31-00014).

Литература

1. Cisco C.V.N.I. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014–2019. [Электронный ресурс]. – URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html (дата обращения 27.05.2015).
2. **Казанский, Н.Л.** Сравнение производительности систем потокового анализа данных в задаче обработки изображений скользящим окном / Н.Л. Казанский, В.И. Проценко, П.Г. Серафимович // Компьютерная оптика. – 2014. – Т. 38, № 4. – С. 804-810.
3. **Казанский, Н.Л.** Распределённая система технического зрения регистрации железнодорожных составов / Н.Л. Казанский, С.Б. Попов // Компьютерная оптика. – 2012. – Т. 36, № 3. – С. 419-428.
4. **Kazanskiy, N.L.** Machine vision system for singularity detection in monitoring the long process / N.L. Kazanskiy, S.B. Popov // Optical Memory and Neural Networks (Information Optics). – 2010. – Vol. 19, No. 1. – P. 23-30.
5. **Зимичев, Е.А.** Пространственная классификация гиперспектральных изображений с использованием метода кластеризации k-means++ / Е.А. Зимичев, Н.Л. Казанский, П.Г. Серафимович // Компьютерная оптика. – 2014. – Т. 38, № 2. – С. 281-286.
6. **Попов, С.Б.** Концепция распределенного хранения и параллельной обработки крупноформатных изображений / С.Б. Попов // Компьютерная оптика. – 2007. – Т. 31, № 4. – С. 77-85.
7. Методы компьютерной оптики / А.В. Волков, Д.Л. Головашкин, Л.Д. Досколович, Н.Л. Казанский, В.В. Котляр, В.С. Павельев, Р.В. Скиданов, В.А. Сойфер, В.С. Соловьев, Г.В. Успленьев, С.И. Харитонов, С.Н. Хонина; под ред. В.А. Сойфера. – Изд. 2-е, испр. – М.: Физматлит, 2003. – 688 с.
8. **Gall, J.** Class-specific hough forests for object detection / J. Gall, V. Lempitsky // Decision forests for computer vision and medical image analysis. – London: Springer, 2013. – P. 143-157.
9. **Cheriyadat, A.M.** Detecting multiple moving objects in crowded environments with coherent motion regions / A.M. Cheriyadat, B.L. Bhaduri, R.J. Radke // Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on. – IEEE, 2008. – P. 1-8.
10. **Felzenszwalb, P.F.** Object detection with discriminatively trained part-based models / P.F. Felzenszwalb, R.B. Girshick, D. Allester, D. Ramanan // Pattern Analysis and Machine Intelligence, IEEE Transactions on. – 2010. – Vol. 32, Issue 9. – P. 1627-1645.
11. **Barinova, O.** On detection of multiple object instances using hough transforms / O. Barinova, V. Lempitsky, P. Kholi // Pattern Analysis and Machine Intelligence, IEEE Transactions on. – 2012. – Vol. 34, Issue 9. – P. 1773-1784.
12. **Kim, S.H.** Real-Time Traffic Video Analysis Using Intel Viewmont Coprocessor / S.H. Kim, J. Shi, Ab. Alfarrarjeh, D. Xu, Yu. Tan, C. Shahabi / Databases in Networked Information Systems. – Springer Berlin Heidelberg, 2013. – P. 150-160.
13. **Bramberger, M.** Real-time video analysis on an embedded smart camera for traffic surveillance / M. Bramberger, J. Brunner, B. Rinner // Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE. – IEEE, 2004. – P. 174-181.
14. **Wójcikowski, M.** FPGA-based real-time implementation of detection algorithm for automatic traffic surveillance sensor network / M. Wójcikowski, R. Żaglewski, B. Pankiewicz // Journal of Signal Processing Systems. – 2012. – Vol. 68, Issue 1. – P. 1-18.
15. **Coifman, B.** A real-time computer vision system for vehicle tracking and traffic surveillance / B. Coifman, D. Beymer, Ph. McLauchlan, J. Malik, J. Malik B // Transportation Research Part C: Emerging Technologies. – 1998. – Vol. 6, Issue 4. – P. 271-288.

16. **Koller, D.** Towards robust automatic traffic scene analysis in real-time / D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, S. Russell // Pattern Recognition, 1994. Vol. 1 – Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on. – IEEE, 1994. – Vol. 1. – P. 126-131.
17. **Bradski, GR.** Computer vision face tracking for use in a perceptual user interface /G.R. Bradski // Applications of Computer Vision, 1998. WACV '98. Proceedings., Fourth IEEE Work-shop on. - 1998. - 214-219.
18. **Guan, P.** Estimating human shape and pose from a single image / P. Guan, Al. Weiss, Al.O. Balan, M.J. Black // Computer Vision, 2009 IEEE 12th International Conference on. – IEEE, 2009. – P. 1381-1388.
19. **Pentland, A.** View-based and modular eigenspaces for face recognition / A. Pentland, B. Moghaddam, T. Starner // Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94, 1994 IEEE Computer Society Conference on. – IEEE, 1994. – P. 84-91.
20. **Ahonen, T.** Face description with local binary patterns: Application to face recognition / T. Ahonen, A. Hadid, M. Pietikainen // Pattern Analysis and Machine Intelligence, IEEE Transactions on. – 2006. – Vol. 28, Issue 12. – P. 2037-2041.
21. **de Araújo, S.A.** Beans quality inspection using correlation-based granulometry / S.A. de Araújo, J.H. Pessota, H.Y. Kim // Engineering Applications of Artificial Intelligence. – 2015. – Vol. 40. – P. 84-94.
22. **Polder, G.** Spectral image analysis for measuring ripeness of tomatoes / G. Polder, G. Van der Heijden, I.T. Young // Transactions-American Society of Agricultural Engineers. – 2002. – Vol. 45, Issue 4. – P. 1155-1162.
23. **Ponsa, D.** Quality control of safety belts by machine vision inspection for real-time production / D. Ponsa, R, Benavente, F. Lumbreras, J. Martı´nez, X. Roca // Optical Engineering. – 2003. – Vol. 42, Issue 4. – P. 1114-1120.
24. **Xie, X.** A review of recent advances in surface defect detection using texture analysis techniques / X. Xie //Electronic Letters on Computer Vision and Image Analysis. – 2008. – Vol. 7, Issue 3. – P. 1-22.
25. **Xu, J.** Vision-guided automatic parking for smart car / J. Xu, G. Chen, M. Xie // Proceedings of the IEEE Intelligent Vehicles Symposium. – 2000. – P. 725-730.
26. **Gavrila, D.M.** Real-time object detection for “smart” vehicles / D.M. Gavrila, V. Philomin // Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on. – IEEE, 1999. – Vol. 1. – P. 87-93.
27. OpenCV Cascade Classification. [Электронный ресурс]. – URL: http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html (дата обращения 20.05.2015).
28. **Viola, P.** Rapid object detection using a boosted cascade of simple features / P. Viola, M. Jones // Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. – IEEE, 2001. – Vol. 1.– P. I-511-I-518.
29. **Lienhart, R.** An extended set of haar-like features for rapid object detection / R. Lienhart, J. Maydt // Image Processing. 2002. Proceedings. 2002 International Conference on. – IEEE, 2002. – T. 1, Vol. 1.– C. I-900-I-903.
30. OpenCV. Haar Cascades. [Электронный ресурс]. – URL: http://alereimondo.no-ip.org/OpenCV/34 (дата обращения 1.06.2015).
31. **Daniels, M.** Real-time human motion detection with distributed smart cameras / M. Daniels, K. Muldawer, J. Schlessman, B. Ozer, W. Wolf // Distributed Smart Cameras, 2007. ICDSC'07. First ACM/IEEE International Conference on. – IEEE, 2007. – P. 187-194.
32. **Sinha, D.** Application of MPI for Efficient Detection and Extraction of Features in Video Surveillance / D. Sinha, G. Sanyal // International Journal of Computer Technology and Applications. – 2011. – Vol. 2, Issue 5. – P. 1269-1274.
33. **Herout, A.** Real-time object detection on CUDA / A. Herout, R. Jošth, R. Juránek, J. Havel, M. Hradiš, P. Zemčík // Journal of Real-Time Image Processing. – 2011. – Vol. 6, Issue 3. – P. 159-170.
34. **Jia, H.** Accelerating viola-jones facce detection algorithm on gpus / H. Jia, Yu. Zhang, W. Wang, J. Xu // High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on. – IEEE, 2012. – P. 396-403.
35. **Norouznezhad, E.** Object tracking on FPGA-based smart cameras using local oriented energy and phase features / Eh. Norouznezhad, Ab. Bigdeli, Ad. Postula, Br.C. Lovell //Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras. – ACM, 2010. – P. 33-40.
36. **Neoh, H.S.** Adaptive edge detection for real-time video processing using FPGAs / H.S. Neoh, A. Hazanchuk // Global Signal Processing. – 2004. – Vol. 7, Issue 3. – P. 2-3.
37. Фреймворк передачи сообщений Netty. [Электронный ресурс]. – URL: http://zeromq.org/ (дата обращения 20.05.2015).
38. Zookeeper. **[**Электронный ресурс**]**. – URL: http://zookeeper.apache.org/ (дата обращения 20.05.2015).
39. **Hirzel, M.** IBM Streams Processing Language: Analyzing Big Data in motion / M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. Mendell, H. Nasgaard, S. Schneider, R. Soule´, K.-L. Wu // IBM Journal of Research and Development. – 2013. – Vol. 57, Issue 3-4. – P. 7:1-7:11. DOI: 10.1147/JRD.2013.2243535.
40. **Gabriela, J.S.** Processing tuples at-least-once in InfoSphere Streams v4 with consistent regions. [Электронный ресурс]. – URL: https://developer.ibm.com/streamsdev/2015/02/20/processing-tuples-least-infosphere-streams-consistent-regions/ (дата обращения 1.06.2015).
41. **Chandy, K.M.** Distributed snapshots: determining global states of distributed systems / K.M. Chandy, L. Lamport // ACM Transactions on Computer Systems (TOCS). – 1985. – Vol. 3, Issue 1. – P. 63-75.
42. **Valles A.** Performance Insights to Intel® Hyper-Threading Technology. [Электронный ресурс]. – URL: https://software.intel.com/en-us/articles/performance-insights-to-intel-hyper-threading-technology?language=es (дата обращения 1.06.2015).
43. CUDA. [Электронный ресурс]. – URL: http://opencv.org/platforms/cuda.html (дата обращения 19.06.2015).

References

[1] Cisco C.V.N.I. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014–2019 White Paper. Source: ⟨http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html⟩.
[2] Kazanskiy NL, Protsenko VI, Serafimovich PG. Comparison of system performance for streaming data analysis in image processing tasks by sliding window. Computer Optics 2014; 38(4): 804-10.
[3] Kazanskiy NL, Popov SB. The distributed vision system of the registration of the railway train. Computer Optics 2012; 36(3): 419-28.
[4] Kazanskiy NL, Popov SB. Machine Vision System for Singularity Detection in Monitoring the Long Process. Op-

tical Memory and Neural Networks (Information Optics) 2010; 19(1): 23-30.
[5] Zimichev EA, Kazanskiy NL. Spectral-spatial classification with k-means++ particional clustering. Computer Optics 2014; 38(2): 281-86.
[6] Popov SB. The concept of distributed storage and parallel processing of large-format images [In Russian]. Computer Optics 2007; 31(4): 77-85.
[7] Gashnikov MV, Glumov NI, Ilyasova NYu, Myasnikov VV, Popov SB, Sergeev VV, Soifer VA, Khramov AG, Chernov AV, Chernov VM, Chicheva MA, Fursov VA. Methods for computer image processing [In Russian]. Ed. Soifer VA. Moscow: "Fizmatlit" Publisher; 2003. 784 p.
[8] Gall J, Lempitsky V. Class-specific hough forests for object detection. Decision forests for computer vision and medical image analysis. Springer London 2013; 143-57.
[9] Cheriyadat AM, Bhaduri BL, Radke RJ. Detecting multiple moving objects in crowded environments with coherent motion regions. Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on. IEEE, 2008; 1-8. DOI: 10.1109/CVPRW.2008.4562983.
[10] Felzenszwalb PF, Girshick RB, Allester D, Ramanan D. Object detection with discriminatively trained part-based models. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2010; 32(9): 1627-45.
[11] Barinova O, Lempitsky V, Kholi P. On detection of multiple object instances using hough transforms. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2012; 34(9): 1773-84.
[12] Kim SH, Shi J, Alfarrarjeh Ab, Xu D, Tan Yu, Shahabi C. Real-Time Traffic Video Analysis Using Intel Viewmont Coprocessor. Databases in Networked Information Systems. Springer Berlin Heidelberg; 2013. 150-60.
[13] Bramberger M, Brunner J, Rinner B. Real-time video analysis on an embedded smart camera for traffic surveillance. Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE. IEEE; 2004. 174-81.
[14] Wójcikowski M, Żaglewski R, Pankiewicz B. FPGA-based real-time implementation of detection algorithm for automatic traffic surveillance sensor network. Journal of Signal Processing Systems. 2012; 68(1): 1-18.
[15] Coifman B, Beymer D, McLauchlan Ph, Malik J, Malik J. B A real-time computer vision system for vehicle tracking and traffic surveillance. Transportation Research Part C: Emerging Technologies. 1998; 6(4): 271-88.
[16] Koller D et al. Towards robust automatic traffic scene analysis in real-time. Pattern Recognition, 1994. Vol. 1 – Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on. IEEE, 1994; 1: 126-31.
[17] Bradski GR. Computer vision face tracking for use in a perceptual user interface. Applications of Computer Vision, 1998, WACV '98, Proceedings, Fourth IEEE Workshop on 1998: 214-19.
[18] Guan P, Weiss Al, Balan AlO, Black MJ Estimating human shape and pose from a single image. Computer Vision, 2009 IEEE 12th International Conference on. IEEE; 2009. 1381-88.
[19] Pentland A, Moghaddam B, Starner T. View-based and modular eigenspaces for face recognition. Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on. IEEE; 1994. 84-91.
[20] Ahonen T, Hadid A, Pietikainen M. Face description with local binary patterns: Application to face recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on. 2006; 28(12): 2037-41.
[21] de Araújo SA, Pessota JH, Kim HY. Beans quality inspection using correlation-based granulometry. Engineering Applications of Artificial Intelligence 2015; 40: 84-94.
[22] Polder G, Van der Heijden G, Young IT. Spectral image analysis for measuring ripeness of tomatoes. Transactions-American Society of Agricultural Engineers 2002; 45(4): 1155-62.
[23] Ponsa D, Benavente R, Lumbreras F, Martı´nez J, Roca X. Quality control of safety belts by machine vision inspection for real-time production. Optical Engineering 2003; 42(4): 1114-20.
[24] Xie X. A review of recent advances in surface defect detection using texture analysis techniques. Electronic Letters on Computer Vision and Image Analysis 2008; 7(3).
[25] Xu J, Chen G, Xie M. Vision-guided automatic parking for smart car. Proceedings of the IEEE Intelligent Vehicles Symposium 2000; 725-30.
[26] Gavrila DM, Philomin V. Real-time object detection for “smart” vehicles. Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on. IEEE, 1999; 1: 87-93.
[27] OpenCV Cascade Classification. Source: ⟨http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html⟩.
[28] Viola P, Jones M. Rapid object detection using a boosted cascade of simple features. Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. IEEE, 2001; 1: I-511-8. DOI: 10.1109/CVPR.2001.990517.
[29] Lienhart R, Maydt J. An extended set of haar-like features for rapid object detection. Image Processing. 2002. Proceedings. 2002 International Conference on. IEEE, 2002; 1: I-900-3.
[30] OpenCV. Haar Cascades. Source: ⟨http://alereimondo.no-ip.org/OpenCV/34⟩.
[31] Daniels M, Muldawer K, Schlessman J, Ozer B, Wolf W. Real-time human motion detection with distributed smart cameras. Distributed Smart Cameras, 2007. ICDSC'07. First ACM/IEEE International Conference on. IEEE, 2007; 187-94.
[32] Sinha D, Sanyal G. Application of MPI for Efficient Detection and Extraction of Features in Video Surveillance. International Journal of Computer Technology and Applications. 2011; 2(5).
[33] Herout A, Jošth R, Juránek R, Havel J, Hradiš M, Zemčík P. Real-time object detection on CUDA. Journal of Real-Time Image Processing. 2011; 6(3): 159-70.
[34] Jia H, Zhang Yu, Wang W, Xu J. Accelerating viola-jones facce detection algorithm on gpus. High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on. IEEE. 2012; 396-403.
[35] Norouznezhad Eh, Bigdeli Ab, Postula Ad, Lovell Br.C. Object tracking on FPGA-based smart cameras using local oriented energy and phase features. Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras. ACM, 2010; 33-40.
[36] Neoh HS, Hazanchuk A. Adaptive edge detection for real-time video processing using FPGAs. Global Signal Processing 2004; 7(3): 2-3.
[37] Message passing framework Netty. Source: ⟨http://zeromq.org/⟩.
[38] Zookeeper. Source: ⟨http://zookeeper.apache.org/⟩.

[39] Hirzel M, Andrade H, Gedik B, Jacques-Silva G, Khandekar R, Kumar V, Mendell M, Nasgaard H, Schneider S, Soule´ R, Wu K.-L. IBM Streams Processing Language: Analyzing Big Data in motion. IBM Journal of Research and Development. 2013; 57(3-4): 7:1-11. DOI: 10.1147/JRD.2013.2243535.

[40] Gabriela JS. Processing tuples at-least-once in InfoSphere Streams v4 with consistent regions. Source: ⟨https://developer.ibm.com/streamsdev/2015/02/20/processing-tuples-least-infosphere-streams-consistent-regions/⟩.

[41] Chandy KM, Lamport L. Distributed snapshots: determining global states of distributed systems. ACM Transactions on Computer Systems (TOCS) 1985; 3(1): 63-75.

[42] Valles A. Performance Insights to Intel® Hyper-Threading Technology. Source: ⟨https://software.intel.com/en-us/articles/performance-insights-to-intel-hyper-threading-technology?language=es⟩.

[43] CUDA platform. Source: ⟨http://opencv.org/platforms/cuda.html⟩.

REAL-TIME ANALYSIS OF PARAMETERS OF MULTIPLE OBJECT DETECTION SYSTEMS

V.I. Protsenko[1], N.L. Kazanskiy[1,2], P.G. Serafimovich[1,2]
[1] Samara State Aerospace University,
[2] Image Processing Systems Institute, Russian Academy of Sciences

Abstract

Analysis of two streaming frameworks: Apache Storm and IBM InfoSphere Streams was performed in solving a multiple object detection task. The analysis focused on two parameters: throughput and 95th percentile of image processing delay. Faces were chosen as the objects to be detected. Profiling was held under CentOS operating systems running on a five node cluster. Face detection was performed using an OpenCV cascade classifier. First architectures and the experiment description were covered. Final suggestions on the applicability of the two systems were made in the concluding section of the article. Apache Storm demonstrated a scalability advantage over IBM InfoSphere Streams in the experiment conducted. It was confirmed that the system based on Apache Storm was able to operate on FullHD video in real-time, achieving throughput of 24 images per second on the hardware used.

Keywords: big data, stream processing, a set of images, a sliding window, used computer memory, image processing, real time system.

Citation: Protsenko VI, Kazanskiy NL, Serafimovich PG. Real-time analysis of parameters of multiple object detection systems. Computer Optics 2015; 39(4): 582-91. DOI: 10.18287/0134-2452-2015-39-4-582-591.

Acknowledgements: The work was funded by the Russian Science Foundation (grant # 14-31-00014).

Сведения об авторах

Проценко Владимир Игоревич, 1991 года рождения. В 2014 году окончил Самарский государственный аэрокосмический университет имени академика С.П. Королёва (СГАУ) со степенью магистра по направлению «Прикладная математика и информатика». Сотрудник НИЛ-97 СГАУ – лаборатории прорывных технологий дистанционного зондирования Земли. Области научных интересов: разработка и исследование программных средств распределённой и параллельной обработки крупноформатных изображений, технологии обработки больших данных.

E-mail: *protsenkovi@gmail.com* .

Vladimir Igorevich Protsenko, Engineer of scientific-research laboratory No 97 of Samara State Aerospace University – laboratory of advanced technologies for remote sensing. His areas of research are parallel and distributed image processing, big data technologies.

*Сведения об авторе **Казанский Николай Львович** – см. стр. 479 этого номера.*

Серафимович Павел Григорьевич, кандидат физико-математических наук; старший научный сотрудник Института систем обработки изображений РАН, докторант Самарского государственного аэрокосмического университета имени академика С.П. Королёва (национального исследовательского университета). Области научных интересов: моделирование и проектирование нанооптических устройств, методы исследования фотонных кристаллов, разработка и исследование программных средств распределённой и параллельной обработки крупноформатных изображений.

E-mail: *serp@smr.ru* .

Pavel Grigorievich Serafimovich, Candidate in Physics and Mathematics; senior researcher at the Image Processing Systems Institute of RAS. His areas of research are nanooptics, simulation and design of photonic crystals, parallel and distributed image processing.

Поступила в редакцию 21 июля 2015 г.
Окончательный вариант – 21 сентября 2015 г.