УДК 004.522

ОБРАБОТКА БОЛЬШИХ ДАННЫХ

Аннаныязова Г.
Преподаватель кафедры «Программное обеспечение информационных технологий»
Институт Телекоммуникаций и информатики Туркменистана,
g.annanyyazova@gmail.com

Аннаныязова Г.
Преподаватель кафедры «Программное обеспечение информационных технологий» Институт Телекоммуникаций и информатики Туркменистана.

Аннотация. Технология обработки *Big Data* сводится к трем основным направлениям, которые решают три типа задач, а именно, перевод и хранение поступаемой информации в гигабайтах, терабайтах, петабайтах и т.д. данных для их обработки, хранения и применения на практике; структурирование разрозненного контента: фотографий, текстов, аудио, видео и всех других видов данных; анализ Больших данных и внедрение разных методов обработки неструктурированных данных, создание разных аналитических отчетов. Чтобы упростить и ускорить работу с большими данными есть много функций языков программирование. На данный момент библиотека *Pandas* является ключевой в анализе данных (*Data Mining*). *Pandas* – это быстрый, мощный, гибкий и простой в использовании инструмент для анализа и обработки данных с открытым исходным кодом, созданный на языке программирования *Python*.

Ключевые слова: Большие данные, обработка, прием данных, сбор данных, анализ данных, представление результатов, *Python*, *Pandas*.

Введение. *Big Data* или большие данные – это структурированные или неструктурированные массивы данных большого объема. Их обрабатывают при помощи специальных автоматизированных инструментов, чтобы использовать для статистики, анализа, прогнозов и принятия решений.

Сам термин «большие данные» предложил редактор журнала *Nature* Клиффорд Линч в спецвыпуске 2008 года. Он говорил о взрывном росте объемов информации в мире. К большим данным Линч отнес любые массивы неоднородных данных более 150 Гб в сутки, однако единого критерия до сих пор не существует. До 2011 года анализом больших данных занимались только в рамках научных и статистических исследований. Но к началу 2012-го объемы данных выросли до огромных масштабов, и возникла потребность в их систематизации и практическом применении.

Большие данные, в свою очередь, становятся источником новых данных, которые стимулируют аналитические инновации в бизнесе, правительстве и академических кругах. Эти нововведения в состоянии радикально изменить взгляд организаций на свой бизнес. Большие данные обеспечат информацию, которая поможет принимать более взвешенные решения, и в некоторых случаях они будут разительно отличаться от тех, что принимаются сегодня. Анализ больших данных даст такое понимание, о котором сегодня

можно только мечтать. Вы увидите, что укрощение волны больших данных и укрощение новых источников данных осуществляется аналогичными способами. Тем не менее дополнительные возможности, которые предоставляют большие данные, требуют использования новейших инструментов, технологий, методов и процессов. [1]

Обработка больших данных. Обработка больших данных в настоящее время с помощью обычных программных методов и аппаратных средств зачастую невозможна, так как этого не позволяет огромный объем имеющейся информации. Несмотря на это, с учетом своих сложностях, несколько эффективных методов для проведения подобных операций все же существует. Для каждого конкретного случая необходимо выбирать наиболее подходящий способ обработки данных, только тогда результаты окажутся удовлетворительными: и с технологической, и с экономической точки зрения.

Если же говорить о терминологии, то «*Big Data*» подразумевает не только данные как таковые, но и принципы обработки больших данных, возможность дальнейшего их использования, порядок обнаружения конкретного информационного блока в больших массивах. Вопросы, связанные с такими процессами, не теряют своей актуальности. Их решение носит важный характер для тех систем, которые многие годы генерировали и копили различную информацию.

Существуют критерии информации, определенные в 2001 году *Meta Group*, которые позволяют оценить, соответствуют ли данные понятию *Big Data* или нет:

– ***Volume*** **(объем)** – примерно 1 Петабайт и выше.

– ***Velocity*** **(скорость)** – генерация, поступление и обработка данных с высокой скоростью.

– ***Variety*** **(разнообразие)** – разнородность данных, различные форматы и возможное отсутствие структурированности.

Архитектура системы обработки. В этом разделе мы будем рассматривать архитектуру системы обработки Больших данных. Для работы с Большими данными используются сложные системы, в которых можно выделить несколько компонентов или слоёв (Layers). Обычно выделяют четыре уровня компонентов таких систем: прием, сбор, анализ данных и представление результатов (рисунок 1).

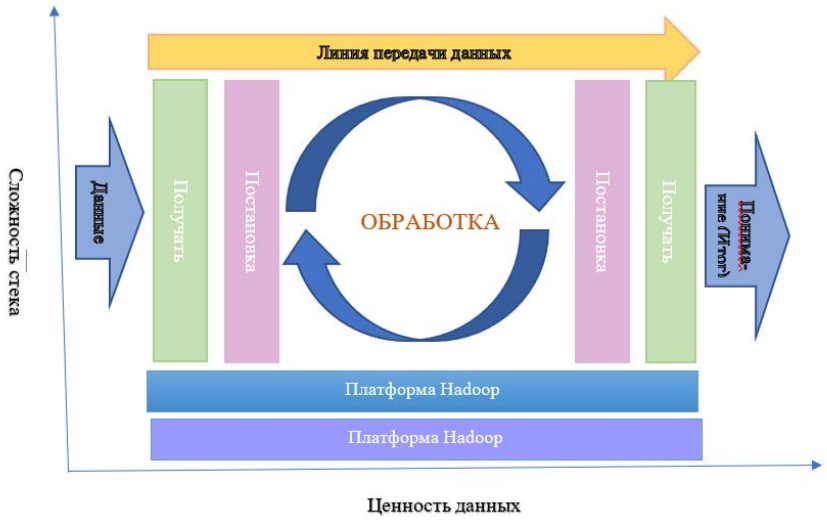

Рисунок 1. Стек работы с Большими данными

Это деление является в значительной мере условным так как, с одной стороны, каждый компонент в свою очередь может быть разделен на подкомпоненты, а с другой некоторые функции компонентов могут перераспределяться в зависимости от решаемой

задачи и используемого программного обеспечения, например, выделяют хранение данных в отдельный слой.

Для работы с Большими данными разработчиками систем создаются модели данных, содержательно связанные с реальным миром. Разработка адекватных моделей данных представляет собой сложную аналитическую задачу, выполняемую системными архитекторами и аналитиками. Модель данных позволяет создать математическую модель взаимодействий объектов реального мира и включает в себя описание структуры данных, методы манипуляции данными и аспекты сохранения целостности данных. Описание разработки моделей данных не является задачей настоящего руководства.

Для хранения данных используются распределенные системы различных типов. Это могут быть файловые системы, базы данных, журналы, механизмы доступа к общей виртуальной памяти. Большинство систем хранения ориентированы исключительно на работу с Большими данными, они имеют крайне ограниченное число функций (например, может отсутствовать возможность не только модификации, но и удаления поступивших данных) что объясняется внутренней сложностью создания высокоэффективных распределенных систем.

Для того, чтобы работа с данными происходила быстрее системы хранения и обработки данных распараллеливаются в кластере (*cluster*, группа компьютеров, объединенных сетью для выполнения единой задачи). Однако, согласно гипотезе Брюера невозможно обеспечить одновременную согласованность (непротиворечивость) данных, доступность данных и устойчивость системы к отделению отдельных узлов. Гипотеза доказана для транзакций типа *ACID* (*Atomic*, *Consistent*. *Isolated*, *Durable*) и известна под названием *CAP* теоремы (*Consistency*, *Availability*, *Partition tolerance*).

Прием данных (*Data Ingestion*). Источники данных имеют различные параметры, такие как частоту поступления данных из источника, объём порции данных, скорость передачи данных, тип поступающих данных и их достоверность.

Для эффективного сбора данных необходимо установить источники данных. Это могут быть хранилища данных, поставщики агрегированных данных, *API* каких-либо датчиков, системные журналы, сгенерированный человеком контент в социальных сетях, в корпоративных информационных системах, геофизическая информация, научная информация, унаследованные данные из других систем.

Источники данных определяют исходный формат данных. Например, мы можем самостоятельно проводить погодные на территории аэропорта, использовать данные, поступающие с взлетающих и садящихся самолетов, закупить данные со спутников, пролетающих над аэропортом и у местной метеослужбы, а также найти их где-то в сети в другом месте. В общем случае для каждого источника необходимо создавать собственный сборщик (*Data Crawler* для сбора информации в сети и *Data Acquisition* для проведения измерений).

Прием данных заключается в начальной подготовке данных от источников с целью приведения данных к общему формату представления данных. Этот единый формат выбирается в соответствии с принятой моделью данных. Выполняются преобразования систем измерения, типов (типизация), верификация. Обработка данных содержательно не затрагивает имеющуюся в данных информацию, но может изменять ее представление (например, приводить координаты к единой системе координат, а значения к единой размерности).

Сбор данных (*Data Staging*). Этап сбора данных характеризуется непосредственным взаимодействием с системами хранения данных. Устанавливается точка сбора, в которой собранные данные снабжаются локальными метаданными и помещаются в хранилище либо передаются для последующей обработки. Данные, по каким-либо причинам не прошедшие точку сбора, игнорируются.

Для структурированных данных проводится преобразование из исходного формата по заранее заданным алгоритмам. Это наиболее эффективная процедура в случае, если структура данных известна. Однако если данные представлены в двоичном виде, структура и связи между данными утеряны, то разработка алгоритмов и основанного на них программного обеспечения для обработки данных может оказаться крайне затруднительной.

Для полуструктурированных данных требуется интерпретация поступающих данных и использование программного обеспечения, умеющего работать с используемым языком описания данных. Существенным плюсом полуструктурированных данных является то, что в них зачастую содержатся не только сами данные, но метаданные в виде информации о связях между данными и способах их получения.

Разработка программного обеспечения для обработки полуструктурированных данных представляет собой достаточно сложную задачу. Однако имеется значительное количество готовых конвертеров, которые могут, например, извлечь данные из формата *XML* в сформированное табличное представление.

Наибольшего объема работ требует обработка неструктурированных данных. Для их перевода к заданному формату может потребоваться создание специального ПО, сложная ручная обработка, распознавание и выборочный ручной контроль.

На этапе сбора проводится контроль типов данных и может выполняться базовый контроль достоверности данных. Например, координаты молекул газа, содержащихся в какой-либо области, не могут лежать за пределами этой области, а скорости – существенно превышать скорость звука. Для того, чтобы избежать ошибок типизации, необходимо проверять правильно ли заданы единицы измерения. Например, в одном наборе данных высота может измеряться в километрах, а в другом – в футах. В этом случае необходимо произвести преобразование высоты в те единицы измерения, которые приняты в используемой модели.

При сборе данные систематизируются и снабжаются метаданными, хранимыми в связанных метаданных. При наличии большого количества источников данных может потребоваться управление сбором данных для того, чтобы сбалансировать объемы информации, поступающие из различных источников.

Собранные данные либо сохраняются в системах хранения, либо (в особенности, для потоковых данных) передаются для анализа в реальном времени.

Анализ данных (*Analysis Layer*). Анализ данных, в отличии от сбора данных, использует информацию, содержащуюся в самих данных. Анализ может проводиться как в реальном времени, так и в пакетном режиме. Анализ данных составляет основную по трудоемкости задачу при работе с Большими данными.

Существует множество методик обработки данных: предиктивный анализ, запросы и отчетность, реконструкция по математической модели, трансляция, аналитическая обработка и другие. Методики используют специфические алгоритмы в зависимости от поставленных целей. Например, аналитическая обработка может являться анализом изображений, социальных сетей, географического местоположения, распознавания по признакам, текстовым анализом, статистической обработкой, анализом голоса, транскрибированием.

Алгоритмы анализа данных также, как и алгоритмы обработки данных, опираются на модель данных. При этом при анализе может быть использовано несколько моделей, задающих общий формат данных, но по-разному моделирующие содержательные процессы, данные о которых мы обрабатываем.

При использовании при анализе методов искусственного интеллекта, в частности нейронных сетей, производится динамическое обучение моделей на различных наборах данных.

При анализе данных производится идентификация сущностей, описываемых данными на основании имеющейся в данных информации и используемых моделей.

Сущностью анализа является аналитических механизм, использующий аналитические алгоритмы, управление моделями и идентификацию сущностей для получения новой содержательной информации, являющейся результатом анализа.

Представление результатов (*Consumption Layer*). Результаты анализа данных предоставляются на уровне потребления. Имеется несколько механизмов, позволяющих использовать результаты анализа больших данных.

1 Мониторинг метаинформации.

Подсистема отображения в реальном времени существенных параметров работы системы, загруженности вычислителей, распределение задач в кластере, распределение информации в хранилищах, наличие свободного места в хранилищах, поступление данных от источников, активности пользователей, отказов оборудования и тд.

2 Мониторинг данных.

Подсистема отображения в реальном времени процессов приема, сбора и анализа данных, навигация по данным.

3 Генерация отчетов, запросы к данным, представление данных в виде визуализации на дэшбордах (*Dashboard*), в формате *PDF*, инфографике, сводных таблицах и кратких справках.

Обработка данных. Чтобы упростить и ускорить работу с большими данными есть много функций языков программирование. Один из первых инструментов, с которым сталкивается аналитик либо *Data Scientist* – это *Pandas*, библиотека *Python* для обработки и анализа данных.

Язык *Python* помогает упростить анализ данных. Если вы научились пользоваться электронными таблицами, то сможете освоить и *pandas*. Несмотря на сходство с табличной компоновкой *Excel, pandas* обладает большей гибкостью и более широкими возможностями. Эта библиотека для *Python* быстро выполняет операции с миллионами строк и способна взаимодействовать с другими инструментами. Она дает идеальную возможность выйти на новый уровень анализа данных. С её помощью мы импортируем и сортируем данные, делаем выборки и находим зависимости. Например, чтобы прочитать файл средствами *Pandas* в *Python* мы пишем:

```
import pandas as pd
data = pd.read_csv('data_file.csv')
```

Такой подход простой и понятный. Каждый *Data Scientist* или аналитик знает это. Но если данных много? Скажем 100 000 000 строк, они постоянно меняются, сроки горят, и до обеда надо проверить еще 100 гипотез?

Возьмем исследовательский набор данных о диабете с сайта *Kaggle* и продублируем каждую строку 100 000 для создания нашего тестового набора данных. В результате получается 76 800 000 строк.

Изначальный датасет выглядит так:

Таблица 1. Набор данных о диабете.

	Pregnancies	***Glucose***	***Blood Pressure***	***Skin Thickness***	***Insulin***	***BMI***	***Diabetes Pedigree Function***	***Age***	***Outcome***
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

Окончание таблицы 1

	Pregnancies	***Glucose***	***Blood Pressure***	***Skin Thickness***	***Insulin***	***BMI***	***Diabetes Pedigree Function***	***Age***	***Outcome***
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

Преобразовываем его для наших экспериментальных задач:

```
# Чтение датасета из файла
df = pd.read_csv('diabetes.csv')

# Создание тестового файла
df = df.loc[df.index.repeat(100000)]

# Сохранение в файл для экспиреметов
df.to_csv('benchmark.csv')

print(len(df), 'строк')
76 800 000 строк
```

Преобразование было долгим, но будем надеется, что время потрачено не зря. Сколько же займет чтение такого файла?

```
df = pd.read_csv('benchmark.csv')
```

Если каждый раз загружать изменившийся датасет заново, на наши 100 гипотез уйдет 13 часов. Сделать до обеда не получится.

Мы хотим загружать данные быстрее, но при этом не терять всех преимуществ, которые даёт *Pandas*. Простая функция использующая *datatable* поможет нам сделать это:

```
import datatable as dt
import pandas as pd
def read_fast_csv(f):
    frame = dt.fread(f)
    ds = frame.to_pandas()
    return ds
```

Попробуем теперь прочитать наш большой датасет:

```
ds = read_fast_csv('benchmark.csv')
```

Это в 6 раз быстрее!

Столбцы с датами в разных форматах. Для анализа и визуализации данных часто требуется работать с датами в разных форматах, когда дата дана только одна. А еще хуже, если все даты прописаны как попало. Приходится приводить все даты к стандартному виду, а потом программным путем добывать из каждой даты разную информацию. Упрощает этот процесс сначала смоделируем ситуацию и создадим функцию, и к нашему

исходному датасету добавим колонку со случайными датами в диапазоне двух последних лет:

```
import numpy as np

# Чтение датасета из файла
df = pd.read_csv('diabetes.csv')
df['date'] = np.random.choice(pd.date_range('2020-01-01', '2022-12-31'), 768)
df.to_csv('diabetes_dates.csv')
```

Ниже код функции, которая к существующему датасету добавляет колонки с наиболее частыми требуемыми форматами, чтобы сделать выборки по годам, по кварталам, месяцам или неделям, либо взять полную дату для целей визуализации.

```
import datetime
from datetime import timedelta

def granular_dates(df, col):
    df['ts_date'] = pd.to_datetime(df[col]).dt.normalize()
                # Полная дата с названием месяца
    df['ts_date_str'] = df["ts_date"].dt.strftime("%d %B %Y")
                # Краткая дата с сокращением месяца
    df['ts_date_str_short'] = df["ts_date"].dt.strftime("%d %b %Y")
    # Только год
    df['ts_year'] = df.ts_date.dt.year
    # Только номер месяца
    df['ts_month'] = df.ts_date.dt.month
    # Только число
    df['ts_day'] = df.ts_date.dt.day
    # Год и квартал
    df['ts_quarter'] = pd.PeriodIndex(df.ts_date, freq="Q").astype(str)
    # Номер недели
    df['ts_dayweek'] = df.ts_date.dt.dayofweek
    # День недели
    df['ts_dayweektext'] = df.ts_date.dt.strftime("%a")
    # Дата конца недели (воскресенья)
    df['ts_week_start'] = df.ts_date.apply(lambda x: x -
timedelta(days=x.weekday())).dt.strftime("%b-%d")
    # Дата конца недели (воскресенья)
    df['ts_week_end'] = df.ts_date.apply(lambda x: x -
timedelta(days=x.weekday()) + timedelta(days=6)).dt.strftime("%b-%d")
```

Теперь всего одна строчка кода (не считая чтение файла):

```
# Чтение датасета
df = pd.read_csv('diabetes_dates.csv')
# Добавление колонок с датами разного формата
granular_dates(df, 'date')
```

Кроме обработке больших данных с помощью *Python* пакета *Pandas*, но также можно достичь этой функциональности при помощи нескольких генераторов. Для обработки больших наборов данных или потоков данных без максимального использования памяти компьютера **создаются конвейеры данных, построенного на генераторах** позволяющие скомпоновать код. Представим, что есть большой *CSV*-файл *big-data.csv* в несколько тысяч строк с данными посещения сайта, которые нужно обработать.

Для обработки такого файла необходимо проделать следующие действия:

1 Прочитать каждую строчку файла.

2 Разбить каждую строку на список значений.
3 Извлечь имена столбцов.
4 Использовать имена столбцов и список значений из строк для создания словарей.
5 Отфильтровать значения, которые не интересуют.
6 Обработать интересующие значения.

Начнем обработку больших данных с чтения каждой строки из *CSV*-файла с помощью выражения генератора. Генераторы ленивы, следовательно. не загружают файл целиком в оперативную память, а читают его построчно.

```
file_name = "big-data.csv"
lines = (line for line in open(file_name))
```

Здесь чтение *CSV*-файла при помощи выражения генератора показано в качестве примера и понимания конвейеров. Для работы с *CSV*-файлами существует специальный оптимизированный модуля *csv,* который включен в стандартную библиотеку *Python*.

Затем будем использовать другое выражение-генератор совместно с предыдущим, чтобы разбить каждую строку на список. Создадим генератор *list_line*, который выполняет итерацию по строкам генератора *lines,* который в свою очередь будет построчно читать большой файл.

```
list_line = (s.rstrip().split(",") for s in lines)
```

По сути - один генератор вкладывается в другой. Это общий шаблон, который используют при проектировании конвейеров генераторов.

Примечание: Код использует метод строки *str.rstrip()* в выражении генератора *list_line,* чтобы убедиться в отсутствии завершающих символов новой строки, которые могут присутствовать в *CSV*-файлах.

Затем извлечем имена столбцов из *big-data.csv.* Так как имена столбцов обычно являются первой строкой *CSV*-файла, то их можно получить при помощи вызова встроенной функции *next():*

```
cols = next(list_line)
```

Вызов функции *next()* сохранит список названий колонок в переменную *cols* и переместит итератор на следующую строку генератора *list_line,* с которой уже начинаются данные.

Чтобы иметь возможность фильтровать и выполнять операции с полученными данными, создадим словари, где ключами будут имена столбцов из *CSV*-файла:

```
log_dicts = (dict(zip(cols, data)) for data in list_line)
```

Выражение-генератора *log_dicts* перебирает списки, созданные генератором *list_line* и использует функции *zip()* и *dict()* для создания словаря, где ключами будут имена столбцов *cols,* а значения – соответствующие данные.

Для обработки данных нужно использовать четвертый генератор, который, например, будет фильтровать данные по столбцу *user_agent*, в котором есть вхождение строки *«YandexBot»* и вытаскивать соответствующие значения столбца *ip_address:*

```
finding = (
      log_dict["ip_address"]
      for log_dict in log_dicts
      if "YandexBot" in log_dict["user_agent"]
)
list_ip = list(
finding)
```

В этом фрагменте кода выражение-генератор *finding* перебирает результаты генератора *log_dicts* и возвращает значение ключа *ip_address* для любого словаря *log_dict,* в котором значение *«YandexBot»* встречается в значении словаря с ключом *user_agent.*

Код выше не перебирает все сразу в выражении генератора *finding.* На самом деле ничего не будет исполняться, пока не будет задействован цикл *for* или функция, которая работает с итерациями, например *list().*

Проще говоря, вызов *list*(*finding*) заставляет работать все генераторы в коде.

И так, собираем код вместе:

```
file_name = "big-data.csv"
# выражение-генератор, который получает строки из файла
lines = (line for line in open(file_name))
# выражение-генератор, который получает список полей из каждой строки
list_line = (s.rstrip().split(",") for s in lines)
# получение списка имен колонок - это первая строка
cols = next(list_line)
# выражение-генератор, создающий словари
log_dicts = (dict(zip(cols, data)) for data in list_line)
# выражение-генератор, фильтрующий нужные значения
finding = (
log_dict["ip_address"]
for log_dict in log_dicts
if "YandexBot" in log_dict["user_agent"]
)
# преобразование генератора в итоговый список
list_ip = list(finding)
```

Этот сценарий объединяет все созданные генераторы, и все они функционируют как один конвейер больших данных.

Заключение. В этой статье мы представили архитектуры обработки больших данных, в которых можно выделить несколько компонентов или слоёв (*Layers*). Детально описано четыре уровня компонентов систем: прием, сбор, анализ данных и представление результатов. А также. для упрощения и ускорение работы с большими данными и их обработки, привели несколько примеров использовав библиотеки Pandas языка программирование Python и создание конвейера данных построенных на генераторах.

Список литературы

[1] Билл Фрэнкс. пер. с англ. Андрея Баранова.Укрощение больших данных: как извлекать смысл из гигантских потоков данных с помощью продвинутой аналитики. — М. : Манн, Иванов и Фербер, 2014.

[2] Радченко И.А, Николаев И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.

PROCESSING BIG DATA

Annanyyazova G.

Lecturer of the Department of Software of information technologies,
The Institute of Telecommunications and Informatics of Turkmenistan

Annotation. Big Data processing technology comes down to three main areas that solve three types of problems, namely, translation and storage of incoming information in gigabytes, terabytes, petabytes, etc. data for their processing, storage and application in practice; structuring disparate content: photographs, texts, audio, video and all other types of data; analysis of Big Data and implementation of various methods for processing unstructured data, creation of various analytical reports. To simplify and speed up working with big data, there are many functions of programming languages. At the moment, the Pandas library is key in data analysis (Data Mining). Pandas is a fast, powerful, flexible, and easy-to-use open-source data science tool built in the Python programming language.

Keywords: Big data, processing, Data Ingestion, Data Staging, Analysis Layer Consumption Layer, Python, Pandas.