Выполнил: Бойко Илья Группа: ПИН-б-о-22-1

Лабораторная работа №9 (Часть 1): Разработка сквозного пайплайна обработки данных

1. Цель работы Разработать комплексное решение для обработки потоковых данных в реальном времени с использованием всего изученного стека технологий. Получить практический опыт построения end-to-end data pipeline.

2. Используемый стек технологий

Платформы: Apache Kafka, Apache Spark, MongoDB Языки программирования: Python (PySpark), JavaScript Библиотеки: kafka-python, pyspark, pymongo, dash/plotly ОС: Linux (Ubuntu) Инструменты: Jupyter Notebook, MongoDB Compass

3. Теоретические сведения

Архитектура пайплайна:

1. **Data Ingestion:** Apache Kafka для приема потоковых данных
2. **Stream Processing:** Apache Spark Structured Streaming для обработки в реальном времени
3. **Data Storage:** MongoDB для хранения результатов обработки
4. **Data Visualization:** Dash/Plotly для визуализации результатов

Ключевые требования:

- Обработка 1000+ сообщений в секунду
- Гарантия доставки сообщений (at-least-once semantics)
- Масштабируемость всех компонентов
- Реальное время визуализации

4. **Ход выполнения работы**

Часть 1: Подготовка инфраструктуры IoT-пайплайна

Обзор пайплайна

IoT Sensors → Kafka Producer → Apache Kafka → Spark Streaming → MongoDB ↑ ↑ ↑ Генерация данных Буферизация Обработка в данных реальном времени

1.1 Развертывание Kafka-кластера

Использование Docker Compose

services: zookeeper: image: confluentinc/cp-zookeeper:7.4.0 container_name: zookeeper environment: ZOOKEEPER_CLIENT_PORT: 2181 ZOOKEEPER_TICK_TIME: 2000 ZOOKEEPER_SYNC_LIMIT: 2 ports: - "2181:2181" healthcheck: test: ["CMD", "bash", "-c", "echo ruok | nc localhost 2181"] interval: 10s timeout: 5s retries: 3 networks: - kafka-network

kafka: image: confluentinc/cp-kafka:7.4.0 container_name: kafka depends_on: zookeeper: condition: service_healthy ports: - "9092:9092" - "29092:29092" environment: KAFKA_BROKER_ID: 1 KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181 KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092 KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1 KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1 KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1 KAFKA_AUTO_CREATE_TOPICS_ENABLE: "true" KAFKA_NUM_PARTITIONS: 3 KAFKA_DEFAULT_REPLICATION_FACTOR: 1 KAFKA_LOG_RETENTION_HOURS: 168 KAFKA_LOG_RETENTION_BYTES: -1 healthcheck: test: ["CMD", "kafka-topics", "--list", "--bootstrap-server", "localhost:9092"] interval: 30s timeout: 10s retries: 3 networks: - kafka-network

kafka-ui: image: provectuslabs/kafka-ui:latest container_name: kafka-ui depends_on: - kafka ports: - "8080:8080" environment: KAFKA_CLUSTERS_0_NAME: local KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: kafka:29092 KAFKA_CLUSTERS_0_ZOOKEEPER: zookeeper:2181 networks: - kafka-network

networks: kafka-network: driver: bridge

Запуск кластера:

docker-compose -f docker-compose-kafka.yml up -d

docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"

docker logs kafka --tail 20

Создание топика:

docker exec kafka kafka-topics.sh --create
--topic sensor-data
--bootstrap-server localhost:9092
--partitions 3
--replication-factor 1
--config retention.ms=604800000
--config segment.bytes=1073741824
--config cleanup.policy=delete

bin/kafka-topics.sh --create
--topic sensor-data

--bootstrap-server localhost:9092
--partitions 3
--replication-factor 1
--config retention.ms=604800000

docker exec kafka kafka-topics.sh --describe --topic sensor-data --bootstrap-server localhost:9092

docker exec kafka kafka-topics.sh --create
--topic test-topic
--bootstrap-server localhost:9092
--partitions 1
--replication-factor 1 Проверка работы Kafka:

echo "Hello, Kafka!" | docker exec -i kafka kafka-console-producer.sh
--topic test-topic
--bootstrap-server localhost:9092

docker exec kafka kafka-console-consumer.sh
--topic test-topic
--bootstrap-server localhost:9092
--from-beginning
--max-messages 1
--timeout-ms 5000

docker exec kafka kafka-topics.sh --list --bootstrap-server localhost:9092

1.2 Настройка MongoDB

Использование Docker Compose

services: mongodb: image: mongo:6.0 container_name: mongodb ports: - "27017:27017" environment: MONGO_INITDB_ROOT_USERNAME: admin MONGO_INITDB_ROOT_PASSWORD: admin123 MONGO_INITDB_DATABASE: iot_analytics volumes: - mongodb_data:/data/db - ./init-mongo.js:/docker-entrypoint-initdb.d/init-mongo.js:ro command: ["--auth"] healthcheck: test: ["CMD", "mongosh", "--eval", "db.adminCommand('ping')"] interval: 10s timeout: 5s retries: 3 networks: - kafka-network

mongo-express: image: mongo-express:latest container_name: mongo-express restart: always ports: - "8081:8081" environment: ME_CONFIG_MONGODB_ADMINUSERNAME: admin ME_CONFIG_MONGODB_ADMINPASSWORD: admin123 ME_CONFIG_MONGODB_SERVER: mongodb ME_CONFIG_BASICAUTH_USERNAME: admin ME_CONFIG_BASICAUTH_PASSWORD: admin123 depends_on: mongodb: condition: service_healthy networks: - kafka-network

volumes: mongodb_data:

networks: kafka-network: external: true name: kafka_kafka-network

Создайте файл инициализации MongoDB:

db = db.getSiblingDB('iot_analytics');

db.createUser({ user: 'iot_app', pwd: 'iot_app_password', roles: [{ role: 'readWrite', db: 'iot_analytics' }, { role: 'read', db: 'admin' }] });

db.createCollection('sensor_metrics'); db.createCollection('anomaly_alerts'); db.createCollection('windowed_metrics');

db.sensor_metrics.createIndex({ timestamp: -1 }); db.sensor_metrics.createIndex({ sensor_id: 1, timestamp: -1 }); db.sensor_metrics.createIndex({ sensor_type: 1, location: 1 });

db.anomaly_alerts.createIndex({ timestamp: -1 }); db.anomaly_alerts.createIndex({ sensor_id: 1, timestamp: -1 }); db.anomaly_alerts.createIndex({ alert_type: 1, timestamp: -1 });

db.windowed_metrics.createIndex({ window_start: -1 }); db.windowed_metrics.createIndex({ sensor_type: 1, location: 1 });

print(' MongoDB инициализирован для IoT Analytics');

Запуск MongoDB:

docker network create kafka_kafka-network 2>/dev/null || true

docker-compose -f docker-compose-mongo.yml up -d

docker ps | grep -E "(mongodb|mongo-express)"

docker logs mongodb --tail 10

Комплексный запуск инфраструктуры:

services: zookeeper: image: confluentinc/cp-zookeeper:7.4.0 container_name: zookeeper environment: ZOOKEEPER_CLIENT_PORT: 2181 ZOOKEEPER_TICK_TIME: 2000 ports: - "2181:2181" networks: - iot-network

kafka: image: confluentinc/cp-kafka:7.4.0 container_name: kafka depends_on: - zookeeper ports: - "9092:9092" - "29092:29092" environment: KAFKA_BROKER_ID: 1 KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181 KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092 KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1 KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1 KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1 networks: - iot-network

mongodb: image: mongo:6.0 container_name: mongodb ports: - "27017:27017" environment: MONGO_INITDB_DATABASE: iot_analytics volumes: - mongodb_data:/data/db - ./init-

mongo.js:/docker-entrypoint-initdb.d/init-mongo.js:ro networks: - iot-network

kafka-ui: image: provectuslabs/kafka-ui:latest container_name: kafka-ui ports: - "8080:8080" environment: KAFKA_CLUSTERS_0_NAME: local KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: kafka:29092 depends_on: - kafka networks: - iot-network

mongo-express: image: mongo-express:latest container_name: mongo-express ports: - "8081:8081" environment: ME_CONFIG_MONGODB_SERVER: mongodb depends_on: - mongodb networks: - iot-network

volumes: mongodb_data:

networks: iot-network: driver: bridge

Запуск всей инфраструктуры:

cat > init-mongo.js << 'EOF' db = db.getSiblingDB('iot_analytics');

db.createCollection('sensor_metrics'); db.createCollection('anomaly_alerts'); db.createCollection('windowed_metrics');

db.sensor_metrics.createIndex({ timestamp: -1 }); db.anomaly_alerts.createIndex({ timestamp: -1 }); db.windowed_metrics.createIndex({ window_start: -1 });

print('MongoDB initialized for IoT Analytics'); EOF

docker-compose up -d

docker-compose ps

docker exec kafka kafka-topics.sh --create
--topic sensor-data
--bootstrap-server localhost:9092
--partitions 3
--replication-factor 1

docker exec kafka kafka-topics.sh --list --bootstrap-server localhost:9092

docker exec mongodb mongosh --eval "db.getMongo().getDBNames()"

Этап 2: Генератор данных для Kafka

Продвинутый продюсер IoT-данных с поддержкой 100+ сообщений/сек 2.1 Оптимизированный производитель с метриками и контролем скорости

import json import time import random import threading from datetime import datetime, timezone from kafka import KafkaProducer from kafka.errors import KafkaError from dataclasses import dataclass, asdict from typing import Dict, List, Tuple import statistics from collections import deque, Counter import logging import signal import sys

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', handlers=[logging.FileHandler('iot_producer.log'), logging.StreamHandler()]) logger = logging.getLogger(**name**)

@dataclass class SensorData: """Модель данных IoT-сенсора""" sensor_id: str sensor_type: str value: float location: str timestamp: str status: str metadata: Dict = None

```
def to_dict(self):
    """Преобразование в словарь для сериализации"""
    data = asdict(self)
    if data['metadata'] is None:
        del data['metadata']
    return data
```

class IoTDataGenerator: """Генератор реалистичных данных IoT-сенсоров"""

```
# Статические данные для генерации реалистичных значений
SENSOR_RANGES = {
    'temperature': {'min': -20.0, 'max': 50.0, 'normal': (15.0, 25.0)},
    'humidity': {'min': 0.0, 'max': 100.0, 'normal': (40.0, 60.0)},
    'pressure': {'min': 950.0, 'max': 1050.0, 'normal': (1000.0, 1020.0)},
    'vibration': {'min': 0.0, 'max': 10.0, 'normal': (0.1, 1.0)},
    'voltage': {'min': 200.0, 'max': 250.0, 'normal': (220.0, 230.0)},
    'current': {'min': 0.0, 'max': 20.0, 'normal': (1.0, 5.0)}
}

LOCATIONS = [
    'factory-floor-a1', 'factory-floor-a2', 'factory-floor-b1',
    'warehouse-north', 'warehouse-south', 'warehouse-east',
    'office-main', 'office-server-room', 'office-lab',
    'outdoor-gate', 'outdoor-roof', 'outdoor-parking'
]

def __init__(self, num_sensors: int = 1000):
    self.num_sensors = num_sensors
    self.sensor_ids = [f"sensor_{i:04d}" for i in range(1, num_sensors + 1)]
    self.sensor_types = list(self.SENSOR_RANGES.keys())

def generate_sensor_data(self, sensor_id: str = None) -> SensorData:
    """Генерация данных одного сенсора"""
    if sensor_id is None:
        sensor_id = random.choice(self.sensor_ids)

    # Определяем тип сенсора на основе ID для консистентности
    sensor_idx = int(sensor_id.split('_')[1]) % len(self.sensor_types)
    sensor_type = self.sensor_types[sensor_idx]

    # Определяем локацию на основе ID
    location_idx = int(sensor_id.split('_')[1]) % len(self.LOCATIONS)
    location = self.LOCATIONS[location_idx]
```

```
    # Генерация реалистичного значения с учетом типа сенсора
    sensor_range = self.SENSOR_RANGES[sensor_type]

    # 95% нормальных значений, 5% аномалий
    if random.random() > 0.05:
        # Нормальное значение (в пределах нормального диапазона)
        value = random.uniform(*sensor_range['normal'])
        status = 'normal'
    else:
        # Аномальное значение (выходящее за пределы)
        if random.random() > 0.5:
            # Слишком высокое значение
            value = random.uniform(sensor_range['normal'][1] * 1.5, sensor_range['max'])
        else:
            # Слишком низкое значение
            value = random.uniform(sensor_range['min'], sensor_range['normal'][0] * 0.5)
        status = 'anomaly'

    # Добавление небольшого шума к значениям
    value += random.uniform(-0.1, 0.1)
    value = round(value, 2)

    # Добавление сезонности (дневной цикл для некоторых типов)
    current_hour = datetime.now(timezone.utc).hour
    if sensor_type == 'temperature':
        # Днем температура выше
        day_factor = 0.5 * abs(12 - current_hour) / 12
        value += random.uniform(-day_factor, day_factor)

    return SensorData(
        sensor_id=sensor_id,
        sensor_type=sensor_type,
        value=value,
        location=location,
        timestamp=datetime.now(timezone.utc).isoformat(),
        status=status,
        metadata={
            'unit': self._get_unit(sensor_type),
            'sensor_model': f"MODEL_{random.randint(1, 5):02d}",
            'battery_level': round(random.uniform(20, 100), 1)
        }
    )

def _get_unit(self, sensor_type: str) -> str:
    """Получение единицы измерения для типа сенсора"""
    units = {
        'temperature': '°C',
        'humidity': '%',
        'pressure': 'hPa',
        'vibration': 'g',
        'voltage': 'V',
        'current': 'A'
```

```
    }
    return units.get(sensor_type, 'unit')
```

class PerformanceMonitor: """Монитор производительности продюсера"""

```
def __init__(self, window_size: int = 100):
    self.window_size = window_size
    self.message_count = 0
    self.error_count = 0
    self.latencies = deque(maxlen=window_size)
    self.start_time = time.time()
    self.type_counter = Counter()
    self.status_counter = Counter()

def record_success(self, latency_ms: float, sensor_type: str, status: str):
    """Запись успешной отправки"""
    self.message_count += 1
    self.latencies.append(latency_ms)
    self.type_counter[sensor_type] += 1
    self.status_counter[status] += 1

def record_error(self):
    """Запись ошибки"""
    self.error_count += 1

def get_metrics(self) -> Dict:
    """Получение текущих метрик"""
    elapsed = time.time() - self.start_time
    if elapsed == 0:
        return {}

    return {
        'total_messages': self.message_count,
        'messages_per_second': self.message_count / elapsed,
        'total_errors': self.error_count,
        'error_rate': self.error_count / max(self.message_count + self.error_count, 1),
        'avg_latency_ms': statistics.mean(self.latencies) if self.latencies else 0,
        'p95_latency_ms': self._percentile(95) if self.latencies else 0,
        'distribution_by_type': dict(self.type_counter),
        'distribution_by_status': dict(self.status_counter)
    }

def _percentile(self, p: float) -> float:
    """Вычисление перцентиля задержек"""
    if not self.latencies:
        return 0
    sorted_latencies = sorted(self.latencies)
    k = (len(sorted_latencies) - 1) * (p / 100)
    f = int(k)
    c = k - f
    return sorted_latencies[f] + c * (sorted_latencies[f + 1] - sorted_latencies[f])
```

class IoTKafkaProducer: """Высокопроизводительный Kafka продюсер для IoT-данных"""

```
def __init__(self, bootstrap_servers: str = 'localhost:9092'):
    self.bootstrap_servers = bootstrap_servers
    self.generator = IoTDataGenerator()
    self.monitor = PerformanceMonitor()
    self.running = False
    self.producer = None

    # Настройка обработки сигналов для graceful shutdown
    signal.signal(signal.SIGINT, self.signal_handler)
    signal.signal(signal.SIGTERM, self.signal_handler)

def initialize_producer(self):
    """Инициализация Kafka продюсера с оптимизированными настройками"""
    try:
        self.producer = KafkaProducer(
            bootstrap_servers=self.bootstrap_servers,
            value_serializer=lambda v: json.dumps(v).encode('utf-8'),
            key_serializer=lambda k: str(k).encode('utf-8') if k else None,

            # Оптимизации для высокой производительности
            acks=1,  # Более производительно чем 'all', но менее надежно
            retries=3,
            compression_type='gzip',  # Сжатие для уменьшения трафика
            linger_ms=5,  # Небольшая задержка для батчинга
            batch_size=16384,  # Размер батча в байтах
            max_in_flight_requests_per_connection=5,  # Параллельные запросы
            buffer_memory=33554432,  # 32MB буфер
            max_block_ms=60000,  # Максимальное время блокировки

            # Настройки сериализации
            key_serializer=lambda k: k.encode('utf-8') if k else None,
            value_serializer=lambda v: json.dumps(v, ensure_ascii=False).encode('utf-8')
        )
        logger.info(f" Kafka producer initialized for {self.bootstrap_servers}")
        return True
    except Exception as e:
        logger.error(f" Failed to initialize Kafka producer: {e}")
        return False

def produce_messages(self, target_rate: int = 100, duration: int = None):
    """Основной цикл генерации и отправки сообщений"""
    if not self.producer:
        if not self.initialize_producer():
            return

    self.running = True
    logger.info(f" Starting message generation at target rate: {target_rate} msg/sec")

    # Расчет интервала между сообщениями
    message_interval = 1.0 / target_rate if target_rate > 0 else 0
```

```
# Статистика для контроля скорости
messages_in_batch = 0
batch_start_time = time.time()

try:
    while self.running:
        batch_start_time = time.time()
        messages_in_batch = 0

        # Генерация и отправка батча сообщений
        while self.running and messages_in_batch < target_rate:
            try:
                start_time = time.perf_counter()

                # Генерация данных сенсора
                sensor_data = self.generator.generate_sensor_data()

                # Отправка в Kafka с ключом для партиционирования
                future = self.producer.send(
                    topic='sensor-data',
                    key=sensor_data.sensor_id,  # Партиционирование по sensor_id
                    value=sensor_data.to_dict()
                )

                # Асинхронная обработка результата
                future.add_callback(
                    lambda metadata, sd=sensor_data, st=start_time:
                    self._on_send_success(metadata, sd, st)
                ).add_errback(
                    lambda error: self._on_send_error(error)
                )

                messages_in_batch += 1

                # Контроль скорости
                if message_interval > 0:
                    time.sleep(message_interval)

            except Exception as e:
                logger.error(f"Error generating/sending message: {e}")
                self.monitor.record_error()

        # Периодический вывод статистики
        if time.time() - batch_start_time >= 10:  # Каждые 10 секунд
            self._print_statistics()

        # Проверка длительности работы
        if duration and time.time() - self.start_time > duration:
            logger.info("Duration limit reached, stopping...")
            break

except KeyboardInterrupt:
    logger.info("Received interrupt signal, stopping...")
```

```
    finally:
        self.shutdown()

def _on_send_success(self, metadata, sensor_data: SensorData, start_time: float):
    """Callback при успешной отправке"""
    latency_ms = (time.perf_counter() - start_time) * 1000
    self.monitor.record_success(latency_ms, sensor_data.sensor_type, sensor_data.status)

    # Логирование только для аномалий (чтобы не засорять логи)
    if sensor_data.status == 'anomaly':
        logger.warning(f" Anomaly detected: {sensor_data.sensor_id} = {sensor_data.value}")

def _on_send_error(self, error: Exception):
    """Callback при ошибке отправки"""
    logger.error(f"Failed to send message: {error}")
    self.monitor.record_error()

def _print_statistics(self):
    """Печать статистики производительности"""
    metrics = self.monitor.get_metrics()

    print("\n" + "=" * 60)
    print(" REAL-TIME PRODUCER STATISTICS")
    print("=" * 60)
    print(f"Total Messages: {metrics.get('total_messages', 0):,}")
    print(f"Messages/sec: {metrics.get('messages_per_second', 0):.1f}")
    print(f"Total Errors: {metrics.get('total_errors', 0)}")
    print(f"Error Rate: {metrics.get('error_rate', 0):.2%}")
    print(f"Avg Latency: {metrics.get('avg_latency_ms', 0):.2f} ms")
    print(f"P95 Latency: {metrics.get('p95_latency_ms', 0):.2f} ms")

    print("\n Distribution by Sensor Type:")
    for sensor_type, count in metrics.get('distribution_by_type', {}).items():
        percentage = count / max(metrics.get('total_messages', 1), 1) * 100
        print(f"  {sensor_type:15} {count:5} ({percentage:5.1f}%)")

    print("\n Anomaly Detection:")
    anomalies = metrics.get('distribution_by_status', {}).get('anomaly', 0)
    total = metrics.get('total_messages', 1)
    print(f"  Anomalies: {anomalies} ({anomalies/total*100:.1f}%)")
    print("=" * 60 + "\n")

def signal_handler(self, signum, frame):
    """Обработчик сигналов для graceful shutdown"""
    logger.info(f"Received signal {signum}, initiating shutdown...")
    self.running = False

def shutdown(self):
    """Корректное завершение работы продюсера"""
    logger.info("Shutting down producer...")
    self.running = False

    if self.producer:
```

```
        # Ожидание отправки всех сообщений
        self.producer.flush(timeout=30)
        self.producer.close(timeout=10)

    # Финальная статистика
    self._print_statistics()
    logger.info("Producer shutdown complete")
```

def run_producer_cli(): """CLI интерфейс для запуска продюсера""" import argparse

```
parser = argparse.ArgumentParser(description='IoT Sensor Data Producer for Kafka')
parser.add_argument('--rate', type=int, default=100,
                    help='Target message rate per second (default: 100)')
parser.add_argument('--duration', type=int, default=None,
                    help='Duration to run in seconds (default: infinite)')
parser.add_argument('--servers', type=str, default='localhost:9092',
                    help='Kafka bootstrap servers (default: localhost:9092)')
parser.add_argument('--sensors', type=int, default=1000,
                    help='Number of unique sensors (default: 1000)')

args = parser.parse_args()

print("\n" + "=" * 60)
print(" IOT SENSOR DATA PRODUCER")
print("=" * 60)
print(f"Target Rate: {args.rate} messages/sec")
print(f"Duration: {args.duration or 'infinite'} seconds")
print(f"Kafka Servers: {args.servers}")
print(f"Unique Sensors: {args.sensors}")
print("=" * 60 + "\n")

producer = IoTKafkaProducer(bootstrap_servers=args.servers)
producer.generator = IoTDataGenerator(num_sensors=args.sensors)

try:
    producer.produce_messages(target_rate=args.rate, duration=args.duration)
except Exception as e:
    logger.error(f"Producer failed: {e}")
    sys.exit(1)
```

if **name** == "**main**": run_producer_cli()