Bases de données et environnements distribués
Chapitre IVb : Intergiciels à Messages (Message Oriented Middleware MOM)
Application avec Apache Kafka

Éric Leclercq

IEM/LE2I

révision. nov. 2015 émail : Eric.Leclercq@u-bourgogne.fr
http://ludique.u-bourgogne.fr/leclercq
http://ufrsciencestech.u-bourgogne.fr

Principes de Kafka

- Kafka est un projet Apache initialement développé par LinkedIn, utilisé par exemple par Airbnb et d'autres grandes entreprises
- Aborde le problème de la communication et de l'intégration de composants dans des systèmes à grande échelle
- Permet d'interconnecter les applications avec un environnement d'analyse de données massives
- Conçu en autre, pour traiter des flux de données d'activités en temps réel (logs, collections d'indicateurs, etc.), haut débit, partitionné
- Kafka est écrit en Scala, et il n'est pas conforme aux spécifications JMS

Spécificités de Kafka

- Supporte un grand nombre de consommateurs (scalability issues)
- Supporte des consommateurs temporaires (ad hoc)
- Supporte des consommateurs en mode batch (par exemple 1 fois par jour en demandant un gros volume de données)
- Haute disponibilité (reprise automatique si un broker disparait)
- Haute performance : plus d'1 million événements par seconde sur un petit cluster
- Polymorphe : messaging system (events), activity tracking (analyse de click sur les applications web), collecte de mesures sur des systèmes avec alertes, audit, stream processing)
- Philosophie : le cluster ne s'occupe pas des clients, il stocke et réparti les messages, pas de transformation de données automatique, pas de cryptage, d'autorisation, d'authentification

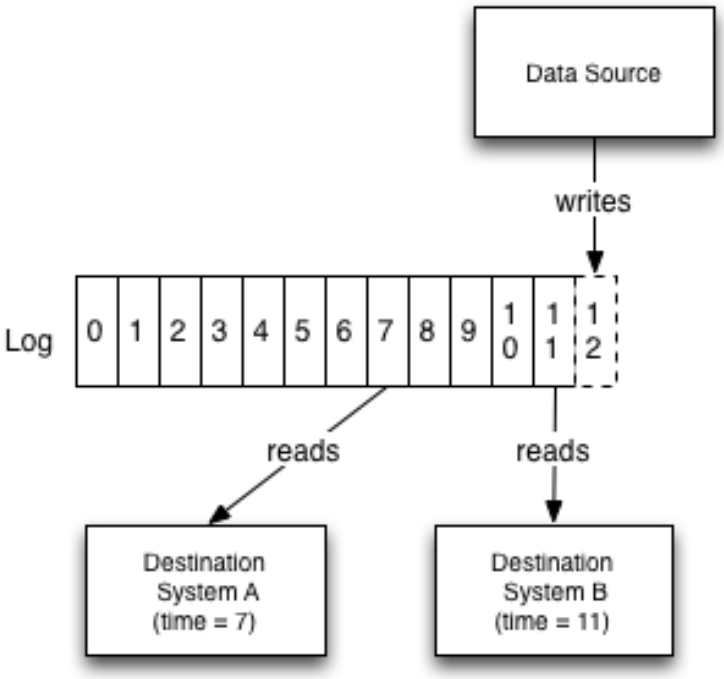
Élements de base

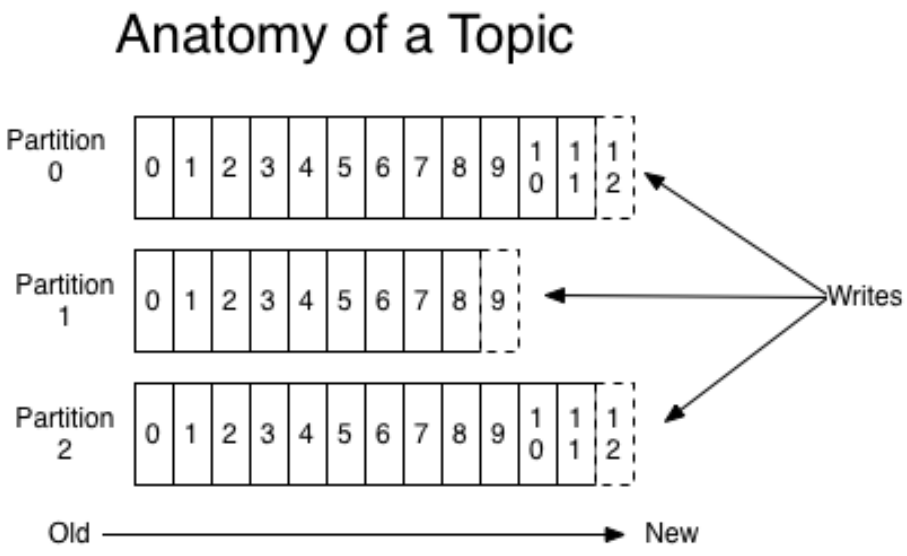
- Les messages sont organisés en Topics
- Mode d'interaction est de type producteurs/consommateurs
- Fonctionne en cluster de brokers (nœuds)
- Les topics volumineux sont répartis dans des partitions sur différents nœuds
- Chaque message à un id, un offset permet de consommer les messages à partir d'un id donné (reprise de données)
- Les partitions sont dupliquées : l'une est le leader (mais les clients n'interagissent pas directement)
- Les figures suivantes sont extraites de la documentation officielle (`http://kafka.apache.org`)

Élements de base

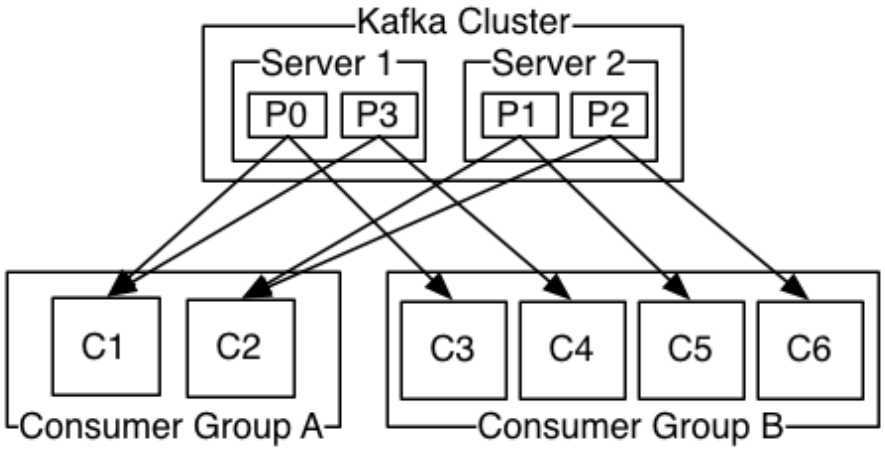
- Un message m est désigné par trois composantes ($Topic$, $Partition$, $offset$)
- Le créateur des topics donne une durée de rétention
- C'est aux consommateurs de se tenir à jour et de gérer leurs lectures
- Les figures suivantes sont extraites de la documentation officielle (`http://kafka.apache.org`)

Élements de base



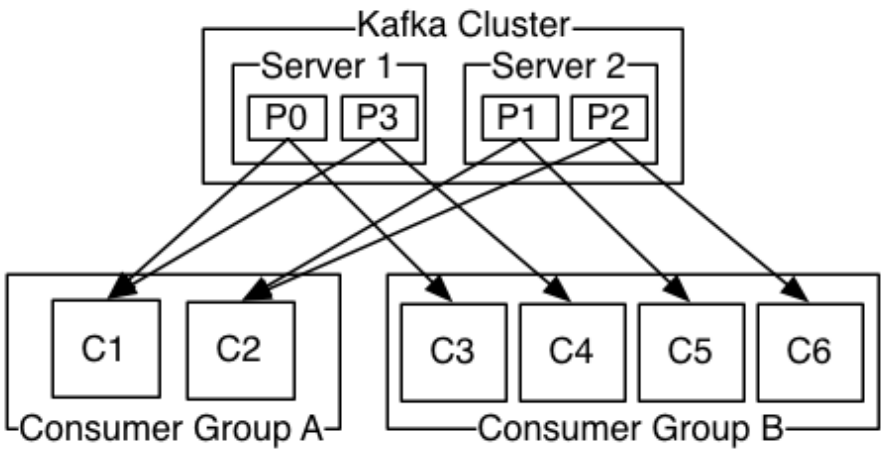


Élements de base et scaling



Élements de base : réplication

Le leader réplique vers les followers. Load balancing entre les partitions Le client détermine à quelle partition il s'adresse



Configuration 2 noeuds, 2 brokers

- Version 0.8.2.2 pour Linux, Scala 2.10, binary version, `http://kafka.apache.org/downloads.html`
- `$ mkdir mykafka`
- `$ cd mykafka`
- `$ cp $HOME/Downloads/kafka_2.10-0.8.2.2.tgz .`
- `$ tar xzfv kafka_2.10-0.8.2.2.tgz`
- Topics et partitions sont écrits dans des *log directories*
  - `$ mkdir kafka-log-1`
  - `$ mkdir kafka-log-2`
- Configurer le serveur kafka
  - `$ cd kafka_2.10-0.8.2.2`
  - `vi config/server.properties`

Configuration du serveur

```
1  # Licensed to the Apache Software Foundation (ASF) under one or more
2  # contributor license agreements.  See the NOTICE file distributed with
3  # this work for additional information regarding copyright ownership.
4  # The ASF licenses this file to You under the Apache License, Version 2.0
5  # (the "License"); you may not use this file except in compliance with
6  # the License.  You may obtain a copy of the License at
7  #
8  #    http://www.apache.org/licenses/LICENSE-2.0
9  #
10 # Unless required by applicable law or agreed to in writing, software
11 # distributed under the License is distributed on an "AS␣IS" BASIS,
12 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 # See the License for the specific language governing permissions and
14 # limitations under the License.
15 # see kafka.server.KafkaConfig for additional details and defaults
16
17 ############################## Server Basics ##############################
18
19 # The id of the broker. This must be set to a unique integer for each broker.
20 broker.id=0
21
22 ############################## Socket Server Settings
       ##############################
23
24 # The port the socket server listens on
25 port=9092
26
27 # Hostname the broker will bind to. If not set, the server will bind to all
       interfaces
28 #host.name=localhost
29 ...
```

Configuration du serveur

```
...
############################# Log Basics #############################
# A comma seperated list of directories under which to store log files
log.dirs=~/mykafka/kafka-log-1

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1

# The number of threads per data directory to be used for log recovery at
    startup and flushing at shutdown.
# This value is recommended to be increased for installations with data dirs
    located in RAID array.
num.recovery.threads.per.data.dir=1
...
############################# Zookeeper #############################
# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000
```

Configuration du serveur

Rôle de ZooKeeper, Kakfa dépend de zookeeper pour :

- maintenir l'état de brokers
- savoir qui est le contrôleur
- connaître le leader
- savoir quels sont les topics etc.
- tous les nœuds doivent référencer le même serveur Zookeeper
- `$ bin/zookeeper-server-start.sh config/zookeeper.properties &`
- `$ bin/kafka-server-start.sh config/server.properties &`
- `$ cp config/server.properties config/server2.properties`
- `$ vi config/server2.properties`
- changer `broker.id`, `port` et `log.dirs`
- `$ bin/kafka-server-start.sh config/server2.properties &`

Création de topic, producer, consumer

- `$ bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic test1 --partitions 2 --replication-factor 2`
- `$ bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic test1`
- `$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test1`
- `$ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test1`
- les consommateurs font référence à Zookeeper pour déterminer les brokers et y enregistrer leurs offsets temporaires

Producteur consommateur en Java

Producteurs ancienne forme :

- Synchrones : sûrs mais pas forcement rapide
- Asynchrones : rapide, sans notification d'erreurs

Nouvelles version des producteurs :

- utilisables de manière synchrone ou asynchrone avec gestion des erreurs incluent des *futures* et des *call_back*
- multi-thread
- l'usage de la mémoire peut être limité/contrôlé

Dans cet exemple le producteur compte et émet les nombres à un intervalle régulier, spécifié.

Producteur

Exemples de codes données sont ceux de Gwen Shapira (https://github.com/gwenshap/kafka-examples).

```
1  import java.util.concurrent.ExecutionException;
2
3  public interface DemoProducer {
4
5      /**
6       * create configuration for the producer
7       * consult Kafka documentation for exact meaning of each configuration
            parameter
8       */
9      public void configure(String brokerList, String sync);
10
11     /* start the producer */
12     public void start();
13
14     /**
15      * create record and send to Kafka
16      * because the key is null, data will be sent to a random partition.
17      * exact behavior will be different depending on producer implementation
18      */
19     public void produce(String s) throws ExecutionException,
           InterruptedException;
20
21     public void close();
22 }
```

Producteur (old style)

```
1  import kafka.javaapi.producer.Producer;
2  import kafka.producer.KeyedMessage;
3  import kafka.producer.ProducerConfig;
4
5  import java.util.Properties;
6
7  // Simple wrapper to the old scala producer, to make the counting code cleaner
8  public class DemoProducerOld implements DemoProducer{
9      private Properties kafkaProps = new Properties();
10     private Producer<String, String> producer;
11     private ProducerConfig config;
12
13     private String topic;
14
15     public DemoProducerOld(String topic) {
16         this.topic = topic;
17     }
18
19     @Override
20     public void configure(String brokerList, String sync) {
21         kafkaProps.put("metadata.broker.list", brokerList);
22         kafkaProps.put("serializer.class", "kafka.serializer.StringEncoder");
23         kafkaProps.put("request.required.acks", "1");
24         kafkaProps.put("producer.type", sync);
25         kafkaProps.put("send.buffer.bytes","550000");
26         kafkaProps.put("receive.buffer.bytes","550000");
27
28         config = new ProducerConfig(kafkaProps);
29     }
```

Producteur (old style)

```
1  @Override
2      public void start() {
3          producer = new Producer<String, String>(config);
4      }
5
6      @Override
7      public void produce(String s) {
8          KeyedMessage<String, String> message = new KeyedMessage<String, String>(
               topic, null, s);
9          producer.send(message);
10     }
11
12     @Override
13     public void close() {
14         producer.close();
15     }
```

Producteur (old style)

```
1  import java.util.concurrent.ExecutionException;
2
3  public class SimpleCounter {
4
5      private static DemoProducer producer;
6
7      public static void main(String[] args) throws InterruptedException,
           ExecutionException {
8
9          if (args.length == 0) {
10             System.out.println("SimpleCounterOldProducer␣{broker-list}␣{topic}␣{
                   type␣old/new}␣{type␣sync/async}␣{delay␣(ms)}␣{count}");
11             return;
12         }
13
14         /* get arguments */
15         String brokerList = args[0];
16         String topic = args[1];
17         String age = args[2];
18         String sync = args[3];
19         int delay = Integer.parseInt(args[4]);
20         int count = Integer.parseInt(args[5]);
21
22         if (age.equals("old"))
23             producer = new DemoProducerOld(topic);
24         else if (age.equals("new"))
25             producer = new DemoProducerNewJava(topic);
26         else
27             System.out.println("Third␣argument␣should␣be␣old␣or␣new,␣got␣" + age
                   );
```

Producteur (old style)

```
1   /* start a producer */
2           producer.configure(brokerList, sync);
3           producer.start();
4
5           long startTime = System.currentTimeMillis();
6           System.out.println("Starting...");
7           producer.produce("Starting...");
8
9           /* produce the numbers */
10          for (int i=0; i < count; i++ ) {
11              producer.produce(Integer.toString(i));
12              Thread.sleep(delay);
13          }
14
15          long endTime = System.currentTimeMillis();
16          System.out.println("... and we are done. This took " + (endTime -
                startTime) + " ms.");
17          producer.produce("... and we are done. This took " + (endTime -
                startTime) + " ms.");
18
19          /* close shop and leave */
20          producer.close();
21          System.exit(0);
22      }
```

Deux mots sur Maven

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
                apache.org/xsd/maven-4.0.0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>ProducerExample</groupId>
8      <artifactId>SimpleCounter</artifactId>
9      <version>1.0-SNAPSHOT</version>
10     <dependencies>
11         <dependency>
12             <groupId>org.apache.kafka</groupId>
13             <artifactId>kafka_2.10</artifactId>
14             <version>0.8.2.2</version>
15         </dependency>
16         <dependency>
17             <groupId>org.apache.zookeeper</groupId>
18             <artifactId>zookeeper</artifactId>
19             <version>3.4.6</version>
20         </dependency>
21     </dependencies>
```

Deux mots sur Maven

```
<build>
  <plugins>
    <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-compiler-plugin</artifactId>
     <configuration>
       <compilerVersion>1.5</compilerVersion>
       <source>1.5</source>
       <target>1.5</target>
     </configuration>
   </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <finalName>uber-${artifactId}-${version}</finalName>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Producteur

Lancer le producteur, ne pas oublier d'avoir lancer un consommateur

- `./run_params.sh localhost:9092 test1 new sync 500 10`

Consommateur

Plusieurs types de consommateurs peuvent être instanciés :

- **High level Consumer :** garder les trace dans Zookeeper de offset lus dans les topics Kafka, loadbalancing automatique
- **Simple Consumer :** API de bas niveau, permet le contrôle de ce qui est lu et des offsets, requiert une expertise
- **New Consumer :** prise en charge automatique des erreurs et du load balancing, gestion manuelle ou automatique des offsets (pas encore dans la version courante)

Consommateur simple

```
1  import kafka.consumer.*;
2  import kafka.javaapi.consumer.ConsumerConnector;
3  import kafka.message.MessageAndMetadata;
4  import kafka.serializer.StringDecoder;
5  import kafka.utils.VerifiableProperties;
6  import org.apache.commons.collections.buffer.CircularFifoBuffer;
7
8  import java.util.HashMap;
9  import java.util.List;
10 import java.util.Map;
11 import java.util.Properties;
12
13 public class SimpleMovingAvgZkConsumer {
14
15     private Properties kafkaProps = new Properties();
16     private ConsumerConnector consumer;
17     private ConsumerConfig config;
18     private KafkaStream<String, String> stream;
19     private String waitTime;
20
21     public static void main(String[] args) {
22         if (args.length == 0) {
23             System.out.println("SimpleMovingAvgZkConsumer␣{zookeeper}␣{group.id}
                   ␣{topic}␣{window-size}␣{wait-time}");
24             return;
25         }
```

Consommateur simple

```
1   String next;
2   int num;
3   SimpleMovingAvgZkConsumer movingAvg = new SimpleMovingAvgZkConsumer();
4   String zkUrl = args[0];
5   String groupId = args[1];
6   String topic = args[2];
7   int window = Integer.parseInt(args[3]);
8   movingAvg.waitTime = args[4];
9   CircularFifoBuffer buffer = new CircularFifoBuffer(window);
10  movingAvg.configure(zkUrl,groupId);
11  movingAvg.start(topic);
12        while ((next = movingAvg.getNextMessage()) != null) {
13            int sum = 0;
14            try {
15                num = Integer.parseInt(next);
16                buffer.add(num);
17            } catch (NumberFormatException e) {
18                // just ignore strings
19            }
20            for (Object o: buffer) {
21                sum += (Integer) o;
22            }
23            if (buffer.size() > 0) {
24                System.out.println("Moving␣avg␣is:␣" + (sum / buffer.size()));
25            }
26            // uncomment if you wish to commit offsets on every message
27            // movingAvg.consumer.commitOffsets();
28        }
29        movingAvg.consumer.shutdown();
30        System.exit(0);
31    }
```

Consommateur simple

```
1   private void configure(String zkUrl, String groupId) {
2         kafkaProps.put("zookeeper.connect", zkUrl);
3         kafkaProps.put("group.id",groupId);
4         kafkaProps.put("auto.commit.interval.ms","1000");
5         kafkaProps.put("auto.offset.reset","largest");
6
7         // un-comment this if you want to commit offsets manually
8         //kafkaProps.put("auto.commit.enable","false");
9
10        // un-comment this if you don't want to wait for data indefinitely
11        kafkaProps.put("consumer.timeout.ms",waitTime);
12
13        config = new ConsumerConfig(kafkaProps);
14    }
```

Consommateur simple

```
1  private void start(String topic) {
2          consumer = Consumer.createJavaConsumerConnector(config);
3          /* We tell Kafka how many threads will read each topic. We have one
                topic and one thread */
4          Map<String, Integer> topicCountMap = new HashMap<String, Integer>();
5          topicCountMap.put(topic,new Integer(1));
6
7          /* We will use a decoder to get Kafka to convert messages to Strings
8          * valid property will be deserializer.encoding with the charset to use.
9          * default is UTF8 which works for us */
10         StringDecoder decoder = new StringDecoder(new VerifiableProperties());
11
12         /* Kafka will give us a list of streams of messages for each topic.
13         In this case, its just one topic with a list of a single stream */
14         stream = consumer.createMessageStreams(topicCountMap, decoder, decoder).
               get(topic).get(0);
15     }
16
17     private String getNextMessage() {
18         ConsumerIterator<String, String> it = stream.iterator();
19         try {
20             return it.next().message();
21         } catch (ConsumerTimeoutException e) {
22             System.out.println("waited␣" + waitTime + "␣and␣no␣messages␣arrived.
                   ");
23             return null;
24         }
25     }
26 }
```

Conclusion : l'eco-système de Kafka

Stream Processing :

- Storm, Samza : stream-processing
- Storm Spout : consomme des messages Kafka et émet des tuples pour Storm
- SparkStreaming : consommateur Kafka pour Spark

Intégration avec Hadoop :

- Flume : collecter et agréger des flux (Kafka Source consumer / Sink producer)
- Camus : initié par LinkedIn, c'est une passerelle Kafka/HDFS pipeline
- Kafka Hadoop Loader

Lambda architecture