Distributed Systems
Coursework

Björn Franke
Overview

• Apache Ignite Framework
  • Overview

• Coursework
  • Getting started
  • Example 1: “Hello, world”
  • Example 2: Word count

• Specification

• Submission Instructions
• Sources of information & examples in this lecture:

  • http://apacheignite.gridgain.org

  • http://ggfabric.blogspot.co.uk

  • https://www.gridgain.com/resources/papers/introducing-apache-ignite

• Note: Piazza online forum now available!
Apache Ignite Overview

• In-memory computing platform

• High-performance transactions

• Real-time streaming

• Fast analytics

• Single, comprehensive data access and processing layer

• Unified API: SQL, C++, .NET, Java/Scala/Groovy, Node.js

• Works on top of existing databases
Apache Ignite Overview

"collection of independent, well-integrated, in-memory components geared to improve performance and scalability of your application"
Cluster Functionality

Cluster functionality is provided via `IgniteCluster` interface. You can get an instance of `IgniteCluster` from `Ignite` as follows:

```java
Ignite ignite = Ignition.ignite();

IgniteCluster cluster = ignite.cluster();
```

Through `IgniteCluster` interface you can:
- Start and stop remote cluster nodes
- Get a list of all cluster members
- Create logical `Cluster Groups`
Data Grid Functionality
Data Streaming Functionality
Compute Grid Functionality

\[ C = C_1 + C_2 + C_3 \]

\[ R = R_1 + R_2 + R_3 \]

\[ \text{IN } T/3 \]
Service Grid Functionality

Service Grid allows for deployments of arbitrary user-defined services on the cluster. You can implement and deploy any service, such as custom counters, ID generators, hierarchical maps, etc. Ignite allows you to control how many instances of your service should be deployed on each cluster node and will automatically ensure proper deployment and fault tolerance of all the services.
Ignite In-Memory Data Fabric, in addition to providing standard key-value map-like storage, also provides an implementation of a fast Distributed Blocking Queue and Distributed Set. `IgniteQueue` and `IgniteSet`, an implementation of `java.util.concurrent.BlockingQueue` and `java.util.Set` interface respectively, also support all operations from `java.util.Collection` interface. Both, queue and set can be created in either collocated or non-collocated mode.

```java
Ignite ignite = Ignition.ignite();

IgniteQueue<String> queue = ignite.queue(
    "queueName", // Queue name.
    0,           // Queue capacity. 0 for unbounded queue.
    null         // Collection configuration.
);
```
Messaging Functionality

Distributed Messaging functionality in Ignite is provided via the `IgniteMessaging` interface. You can get an instance of `IgniteMessaging`, like so:

```java
Ignite ignite = Ignition.ignite();

IgniteMessaging rmtMsg = ignite.message(ignite.cluster().forRemotes());

// Add listener for unordered messages on all remote nodes.
rmtMsg.remoteListen("MyOrderedTopic", (nodeId, msg) -> {
    System.out.println("Received ordered message [msg=" + msg + ", from=" + nodeId + "]");

    return true; // Return true to continue listening.
});

// Send ordered messages to remote nodes.
for (int i = 0; i < 10; i++)
    rmtMsg.sendOrdered("MyOrderedTopic", Integer.toString(i));
```
Distributed Events Functionality

Ignite ignite = Ignition.ignite();

// Local listener that listens to local events.
IgnitePredicate<CacheEvent> locLsnr = evt -> {
    System.out.println("Received event [evt=" + evt.name() + ", key=" + evt.key() + ", oldVal=" + evt.oldValue() + ", newVal=" + evt.newValue());

    return true; // Continue listening.
};

// Subscribe to specified cache events occurring on local node.
ignite.events().localListen(locLsnr,
    EventType.EVT_CACHE_OBJECT_PUT,
    EventType.EVT_CACHE_OBJECT_READ,
    EventType.EVT_CACHE_OBJECT_REMOVED);

// Get an instance of named cache.
final IgniteCache<Integer, String> cache = ignite.cache("cacheName");

// Generate cache events.
for (int i = 0; i < 20; i++)
    cache.put(i, Integer.toString(i));
Getting started

1. Download and Install Ignite
   Download the latest binary distribution from the Apache Ignite website and extract the resulting .zip file to a location of your choice:

   $ unzip apache-ignite-fabric-1.7.0-bin.zip
   $ cd apache-ignite-fabric-1.7.0-bin

2. Set Environment Variable (this step is optional)
   Set IGNITE_HOME environment variable to point to the installation folder and make sure there is no trailing / in the path. On my Mac, I have set this environment variable in .bash_profile file, like so:

   export IGNITE_HOME=<path-to-ignite-installation-folder>

3. Start Ignite Cluster
   Start a node using bin/ignite.sh command and specify an example configuration file provided in the Ignite installation:

   $ bin/ignite.sh examples/config/example-ignite.xml
   If the installation was successful, your Ignite node startup message should look like this:
Getting started

```
~/apache-ignite-fabric-1.5.0-b1-bin$ bin/ignite.sh examples/config/example-ignite.xml
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=256m; support was removed in 8.0
[09:45:14]                                                                                       /
[09:45:14]                                                                                       /
[09:45:14]                                                                                       /
[09:45:14] ver. 1.5.0-b1#20151201-sha1:062d440c
[09:45:14] 2015 Copyright(C) Apache Software Foundation
[09:45:14] Ignite documentation: http://ignite.apache.org
[09:45:14] Quiet mode.
[09:45:14] ^-- Logging to file '/Users/prachig/apache-ignite-fabric-1.5.0-b1-bin/work/log/ignite-977597db.0.log'
[09:45:14] ^-- To see **FULL** console log here add -DIGNITE_QUIET=false or "-v" to ignite.sh
[09:45:14] OS: Mac OS X 10.10.5 x86_64
[09:45:14] VM information: Java(TM) SE Runtime Environment 1.8.0_31-b13 Oracle Corporation Java
HotSpot(TM) 64-Bit Server VM 25.31-b07
[09:45:14] Configured plugins:
[09:45:14] ^-- None
[09:45:14] Security status [authentication=off, tls/ssl=off]
[09:45:15] Performance suggestions for grid (fix if possible)
[09:45:15] To disable, set -DIGNITE_PERFORMANCE_SUGGESTIONS_DISABLED=true
[09:45:15] ^-- Disable peer class loading (set 'peerClassLoadingEnabled' to false)
[09:45:15] ^-- Disable grid events (remove 'includeEventTypes' from configuration)
[09:45:15] To start Console Management & Monitoring run Ignitvisorcmd.{sh|bat}
[09:45:15] Ignite node started OK (id=977597db)
[09:45:15] Topology snapshot [ver=1, servers=1, clients=0, CPUs=4, heap=1.0GB]
```
Getting started

I have started one more node in another terminal, by repeating the above command (in step 3). I now have an Ignite cluster setup with two server nodes running. You can start as many nodes as you like. Ignite will automatically discover all the nodes.
Getting started

4. Add Ignite Dependency (only if you are using Maven)
   Add the following Ignite dependencies in your project’s `pom.xml` file:

   ```xml
   <dependency>
     <groupId>org.apache.ignite</groupId>
     <artifactId>ignite-core</artifactId>
     <version>1.5.0-b1</version>
   </dependency>

   <dependency>
     <groupId>org.apache.ignite</groupId>
     <artifactId>ignite-spring</artifactId>
     <version>1.5.0-b1</version>
   </dependency>
   ``

“Hello world” Example

Here is a sample **HelloWord.java** file that prints ‘Hello World’ on all the nodes in the cluster.

```java
import org.apache.ignite.Ignite;
import org.apache.ignite.IgniteCache;
import org.apache.ignite.IgniteException;
import org.apache.ignite.Ignition;

public class HelloWorld {
    public static void main(String[] args) throws IgniteException {
        try (Ignite ignite = Ignition.start("examples/config/example-ignite.xml")) {
            // Put values in cache.
            IgniteCache<Integer, String> cache = ignite.getOrCreateCache("myCache");

            cache.put(1, "Hello");
            cache.put(2, "World!");

            // Get values from cache and
            // broadcast 'Hello World' on all the nodes in the cluster.
            ignite.compute().broadcast(() -> {
                String hello = cache.get(1);
                String world = cache.get(2);

                System.out.println(hello + " " + world);
            });
        }
    }
}
```
"Hello world" Example
“Hello world” Example

Demo!
Word Count Example

In this example we will stream text into Ignite and count each individual word. We will also issue periodic SQL queries into the stream to query top 10 most popular words.

The example will work as follows:
1. We will setup up a cache to hold the words and their counts.
2. We will setup a 5 second sliding window to keep the word counts only for last 5 seconds.
3. \texttt{StreamWords} program will stream text data into Ignite.
4. \texttt{QueryWords} program will query top 10 words out of the stream.
We define a `CacheConfig` class which will provide configuration to be used from both programs, `StreamWords` and `QueryWords`. The cache will use words as keys, and counts for words as values. Note that in this example we use a sliding window of 5 seconds for our cache. This means that words will disappear from cache after 5 seconds since they were first entered into cache.

```java
public class CacheConfig {
    public static CacheConfiguration<String, Long> wordCache() {
        CacheConfiguration<String, Long> cfg = new CacheConfiguration<>("words");

        // Index the words and their counts,
        // so we can use them for fast SQL querying.
        cfg.setIndexedTypes(String.class, Long.class);

        // Sliding window of 5 seconds.
        cfg.setExpiryPolicyFactory(FactoryBuilder.factoryOf(
            new CreatedExpiryPolicy(new Duration(SECONDS, 5))));

        return cfg;
    }
}
```
We define a `StreamWords` class which will be responsible to continuously read words from a local text file ("alice-in-wonderland.txt" in our case) and stream them into Ignite "words" cache.

1. We set `allowOverwrite` flag to `true` to make sure that existing counts can be updated.
2. We configure a `StreamTransformer` which takes currently cache count for a word and increments it by 1.
public class StreamWords {
    public static void main(String[] args) throws Exception {
        // Mark this cluster member as client.
        Ignition.setClientMode(true);

        try (Ignite ignite = Ignition.start()) {
            IgniteCache<String, Long> stmCache =
                ignite.getOrCreateCache(CacheConfig.wordCache);

            // Create a streamer to stream words into
            // the cache.
            try (IgniteDataStreamer<String, Long> stmr =
                 ignite.dataStreamer(stmCache.getName())) {
                // Allow data updates.
                stmr.allowOverwrite(true);

                // Configure data transformation to count
                instances of the same word.
                stmr.receiver(StreamTransformer.from((e, arg) ->
                {
                    // Get current count.
                    Long val = e.getValue();

                    // Increment current count by 1.
                    e.setValue(val == null ? 1L : val + 1);

                    return null;
                }));
            }

            // Stream words from "alice-in-wonderland"
            while (true) {
                Path path =
                    Paths.get(StreamWords.class.getResource("alice-
                        in-wonderland.txt").toURI());

                // Read words from a text file.
                try (Stream<String> lines =
                     Files.lines(path)) {
                    lines.forEach(line -> {
                        Stream<String> words =
                            Stream.of(line.split(" "));

                        // Stream words into Ignite streamer.
                        words.forEach(word -> {
                            if (!word.trim().isEmpty())
                                stmr.addData(word, 1L);
                        });
                    });
                }
            }
        }
    }
}
QueryWords class

We define a `QueryWords` class which will periodically query word counts from the cache.

1. We use standard SQL to query the counts.
2. Ignite SQL treats Java classes as SQL tables. Since our counts are stored as simple `Long` type, the SQL query below queries `Long` table.
3. Ignite always stores cache keys and values as `_key` and `_val` fields, so we use this syntax in our SQL query.
public class QueryWords {
    public static void main(String[] args) throws Exception {
        // Mark this cluster member as client.
        Ignition.setClientMode(true);

        try (Ignite ignite = Ignition.start()) {
            IgniteCache<String, Long> stmCache = ignite.getOrCreateCache(CacheConfig.wordCache);

            // Select top 10 words.
            SqlFieldsQuery top10Qry = new SqlFieldsQuery(
                "select _key, _val from Long order by _val desc limit 10");

            // Query top 10 popular words every 5 seconds.
            while (true) {
                // Execute queries.
                List<List<?>> top10 = stmCache.query(top10Qry).getAll();

                // Print top 10 words.
                ExamplesUtils.printQueryResults(top10);

                Thread.sleep(5000);
            }
        }
    }
}
Coursework Specification

• Find top ten most accessed Wikipedia pages from page access log files

• Part A: Sequential, non-distributed
  Using Ignite stream log files and process them in a cache, applying a 1-second sliding window

• Part B: Concurrently, distributed
  Using Ignite stream these log files, and process them concurrently. Determine global top ten most popular pages on Wikibooks

• Details in hand-out, which will be uploaded to the course website shortly

• Submission date: 4pm on Thursday, November 17, 2016