Apache Beam (incubating) Proposal: Part 1: IOChannelFactory Redesign

Author: Pei He
JIRA issue: BEAM-59
Last Updated: Dec 2, 2016

Goals & Motivation

This proposal is to support file-based IOs (TextIO, AvroIO) with user-defined file system, such as hdfs, s3, azure.

Goal: Design an API interface to interact with a file system (rename, delete, copy, stat...).
Non-goal: Unify formats of file-based IOs (newline delimited, avro).

Scope & Impact

The scope of this proposal is in the core sdk. Users include:

1. SDK library writers could use the proposed interfaces to make their code file-system-independent.
2. SDK library writers could implement the proposed interfaces to support additional file systems.

API Proposal

Current IOChannelFactory API to be replaced:

```
public interface IOChannelFactory {
  Collection<String> match(String spec) throws IOException;
  ReadableByteChannel open(String spec) throws IOException;
  WritableByteChannel create(String spec, String mimeType) throws IOException;
  long getSizeBytes(String spec) throws IOException;
  boolean isReadSeekEfficient(String spec) throws IOException;
  public String resolve(String path, String other) throws IOException;
}
```

Proposed API:

Noticeable proposed changes:

1. FileSystem interface for providers to implement, and FileSystems utility for clients.
2. FileSystems utility includes the options parameter in most methods to specify behaviors.
3. ~~Replace String with URI to include scheme for files/directories locations.~~
4. Require file systems to provide a SeekableByteChannel for read.
5. Additional methods, such as rename().

Section 1: interface for file systems providers.

```
/**
 * File system interface in Beam.
 *
 * <p>New file systems should implement {@link FileSystem}, and clients should
 * use {@link FileSystems} utility.
 */
public interface org.apache.beam.sdk.io.FileSystem {

  // 1. Create write channel.
  // Contract: Create() makes directories if necessary.
  protected WritableByteChannel create(String file, CreateOptions options)
      throws IOException;

  class CreateOptions {
    public String mimeType();
    // Other possible properties: overwrite(), bufferSize(), blockSize().
  }

  // 2. Open read channel.
  protected ~~SeekableByteChannel~~ ReadableByteChannel open(String file) throws
IOException;

  // 3. Rename files.
  /**
   * Contracts:
   * 1. srcFiles have to exist.
   * 2. destFiles will be created recursively.
   * 3. When rename throws, the state of the files is unknown but safe:  for every
<source,dest> pair of files, the following are possible: a) source exists, b) dest
exists, c) source and dest both exist. Thus no data is lost, however,
   * duplicated files are possible -- some files might be copied, some might not.
   * In failure scenarios, callers can use {@link #match} [e.g.] to determine the
   * state of the files.
   */
```

```
  protected void rename(List<String> srcFiles, List<String> destFiles) throws
IOException;

  // 4. Delete files
  /**
   * Contracts:
   * 1. Throws FileNotFoundException if files are missing.
   * 2. It is recommended not to delete recursively, and let clients to depend on
   * FileSystems' utility function. However, recursively deletion is not considered
   * as an error.
   * 3. When delete throws, each file might or might not be deleted.
   * In such cases, callers can use {@link #match} to determine the state of the
   * files.
   */
  protected void delete(Collection<String> files) throws IOException;

  // This is removed since URI supports resolve and parse file names.
  [FilePath toPath(URI path); ]

  // isGlob() is removed since we can use match() to handle the ambiguities of
  // user strings, see {@link #match()}.
  [boolean isGlob(URI spec);]

  // getMetadata() is removed since we can use match() to get file metadata.
  [protected List<Metadata> getMetadata(Collection<URI> uris) throws IOException;]

  // 5. match
  /**
   * This is the entry point to convert users provided specs to URIs.
   * Callers should use match() to resolve users specs ambiguities before
   * calling other methods.
   *
   * @return List<MatchResult>, which is in the same order of the input specs.
   *
   * @throw IOException if all specs failed to match due to issues like:
   * network connection, authorization.
   * Exception for individual spec need to be deferred until callers retrieve
   * metadata with {@link MatchResult#metadata()}.
   *
   *
   * Implementation should handle the following ambiguities of users provided sepc:
   * 1). spec could be a glob or a file. match() should be able to tell and
   *     choose efficient implementations.
   * 2). spec doesn't end with '/' may refer to files or directories:
   *     file:/home/dir/ should be returned for spec "file:/home/dir".
   *     (However, spec ends with '/' always refers to directories.)
   * Note: File systems glob support is different. However, it is required to
```

```
   * support glob in the final component of a path (eg file:/foo/bar/*.txt).
   *
   * Throws if the spec is invalid.
   */
  protected List<Metadata> match(String spec);

  protected List<MatchResult> match(List<String> specs)

  class MatchResult {
    public Status status();
    /**
     * @throws the exception for the corresponding spec, if the status() is not OK.
     */
    public Metadata[] metadata() throws Exception;

    class Metadata {
      /**
       * Returns the absolute path.
       *
       * Directories URIs should end with '/' in order for URI.resolve() to
       * work correctly.
       */
      public String path();
      public Long sizeBytes();
      public Boolean isDirectory();
      public Boolean isReadSeekEfficient();
    }

    enum Status {
      OK,
      NOT_FOUND,
      ERROR,
    }
  }
}
```

Section 2: Utility class for clients.

```
/**
 * Clients facing utility functions for file system operations.
 */
public class FileSystems {
  public static WritableByteChannel create(String file, CreateOptions options)
      throws IOException;

  public static SeekableByteChannel open(String file, OpenOptions options)
      throws IOException;
```

```
  class OpenOptions {
    public long startPosition();
    // Other possible properties: bufferSize(), blockSize().
  }

  public static void delete(Collection<String> files, DeleteOptions options);

  class DeleteOptions {
    // Matches with the prefix, and deletes level by level from the leaf to the
root.
    public boolean recursive();
    // Calls getMetadata() and deletes existing files.
    public boolean ignoreMissingFile();
  }

  public static void rename(
      List<String> srcfiles,
      List<String> destfiles,
      RenameOptions options) throws IOException;

  class RenameOptions {
    public boolean ignoreMissingFile();
    // Other possible properties: overwrite()
  }

  public static List<Metadata> match(URI glob, MatchOptions options);

  public static List<MatchResult> match(List<String> specs, MatchOptions options);

  class MatchOptions {
    // Choose to filter directories based on Metadata.isDirectory().
    boolean includeDir();
  }
}
```

Section 3: FileSystem specific operation options.

```
public class GcsFileSystem extends FileSystem {
  @Override protected WritableByteChannel create(
      String file, CreateOptions createOptions) throws IOException {
    if (createOptions instanceof GcsCreateOptions) {
      ...
    } else {
      ...
    }
  }
  public static class GcsCreateOptions extends CreateOptions {
```

```
    public int uploadBufferSizeBytes() {};
  }
}
```