
# Pós-Graduação em Computação Distribuída e Ubíqua

## INF628 - Engenharia de Software para Sistemas Distribuídos
## Design Patterns para MapReduce (parte II)

Sandro S. Andrade
sandroandrade@ifba.edu.br

# Design Patterns para MR

- Patterns para joins
  - Exemplo:

*Table 5-1. Table A*

| User ID | Reputation | Location |
|---|---|---|
| 3 | 3738 | New York, NY |
| 4 | 12946 | New York, NY |
| 5 | 17556 | San Diego, CA |
| 9 | 3443 | Oakland, CA |

*Table 5-2. Table B*

| User ID | Post ID | Text |
|---|---|---|
| 3 | 35314 | Not sure why this is getting downvoted. |
| 3 | 48002 | Hehe, of course, it's all true! |
| 5 | 44921 | Please see my post below. |
| 5 | 44920 | Thank you very much for your reply. |
| 8 | 48675 | HTML is not a subset of XML! |

# Design Patterns para MR

- Patterns para joins
  - Inner Join:

| A.User ID | A.Reputation | A.Location | B.User ID | B.Post ID | B.Text |
|---|---|---|---|---|---|
| 3 | 3738 | New York, NY | 3 | 35314 | Not sure why this is getting downvoted. |
| 3 | 3738 | New York, NY | 3 | 48002 | Hehe, of course, it's all true! |
| 5 | 17556 | San Diego, CA | 5 | 44921 | Please see my post below. |
| 5 | 17556 | San Diego, CA | 5 | 44920 | Thank you very much for your reply. |

# Design Patterns para MR

- Patterns para joins
  - Left Outer Join:

| A.User ID | A.Reputation | A.Location | B.User ID | B.Post ID | B.Text |
|---|---|---|---|---|---|
| 3 | 3738 | New York, NY | 3 | 35314 | Not sure why this is getting downvoted. |
| 3 | 3738 | New York, NY | 3 | 48002 | Hehe, of course, it's all true! |
| 4 | 12946 | New York, NY | null | null | null |
| 5 | 17556 | San Diego, CA | 5 | 44921 | Please see my post below. |
| 5 | 17556 | San Diego, CA | 5 | 44920 | Thank you very much for your reply. |
| 9 | 3443 | Oakland, CA | null | null | null |

# Design Patterns para MR

- Patterns para joins
  - Right Outer Join:

| A.User ID | A.Reputation | A.Location | B.User ID | B.Post ID | B.Text |
|---|---|---|---|---|---|
| 3 | 3738 | New York, NY | 3 | 35314 | Not sure why this is getting downvoted. |
| 3 | 3738 | New York, NY | 3 | 48002 | Hehe, of course, it's all true! |
| 5 | 17556 | San Diego, CA | 5 | 44921 | Please see my post below. |
| 5 | 17556 | San Diego, CA | 5 | 44920 | Thank you very much for your reply. |
| null | null | null | 8 | 48675 | HTML is not a subset of XML! |

# Design Patterns para MR

- Patterns para joins
  - Full Outer Join:

| A.User ID | A.Reputation | A.Location | B.User ID | B.Post ID | B.Text |
|---|---|---|---|---|---|
| 3 | 3738 | New York, NY | 3 | 35314 | Not sure why this is getting downvoted. |
| 3 | 3738 | New York, NY | 3 | 48002 | Hehe, of course, it's all true! |
| 4 | 12946 | New York, NY | null | null | null |
| 5 | 17556 | San Diego, CA | 5 | 44921 | Please see my post below. |
| 5 | 17556 | San Diego, CA | 5 | 44920 | Thank you very much for your reply. |
| null | null | null | 8 | 48675 | HTML is not a subset of XML! |
| 9 | 3443 | Oakland, CA | null | null | null |

# Design Patterns para MR

- Patterns para joins
  - Antijoin:

| A.User ID | A.Reputation | A.Location | B.User ID | B.Post ID | B.Text |
|---|---|---|---|---|---|
| 4 | 12946 | New York, NY | null | null | null |
| null | null | null | 8 | 48675 | HTML is not a subset of XML! |
| 9 | 3443 | Oakland, CA | null | null | null |

# Design Patterns para MR


- Patterns para joins
  - Produto Cartesiano:

| A.User ID | A.Reputation | A.Location | B.User ID | B.Post ID | B.Text |
|---|---|---|---|---|---|
| 3 | 3738 | New York, NY | 3 | 35314 | Not sure why this is getting downvoted. |
| 3 | 3738 | New York, NY | 3 | 48002 | Hehe, of course, it's all true! |
| 3 | 3738 | New York, NY | 5 | 44921 | Please see my post below. |
| 3 | 3738 | New York, NY | 5 | 44920 | Thank you very much for your reply. |
| 3 | 3738 | New York, NY | 8 | 48675 | HTML is not a subset of XML! |
| 4 | 12946 | New York, NY | 3 | 35314 | Not sure why this is getting downvoted. |
| 4 | 12946 | New York, NY | 3 | 48002 | Hehe, of course, it's all true! |
| 4 | 12946 | New York, NY | 5 | 44921 | Please see my post below. |
| 4 | 12946 | New York, NY | 5 | 44920 | Thank you very much for your reply. |
| 4 | 12946 | New York, NY | 8 | 48675 | HTML is not a subset of XML! |
| 5 | 17556 | San Diego, CA | 3 | 35314 | Not sure why this is getting downvoted. |
| 5 | 17556 | San Diego, CA | 3 | 48002 | Hehe, of course, it's all true! |
| 5 | 17556 | San Diego, CA | 5 | 44921 | Please see my post below. |
| 5 | 17556 | San Diego, CA | 5 | 44920 | Thank you very much for your reply. |
| 5 | 17556 | San Diego, CA | 8 | 48675 | HTML is not a subset of XML! |
| 9 | 3443 | Oakland, CA | 3 | 35314 | Not sure why this is getting downvoted. |
| 9 | 3443 | Oakland, CA | 3 | 48002 | Hehe, of course, it's all true! |
| 9 | 3443 | Oakland, CA | 5 | 44921 | Please see my post below. |
| 9 | 3443 | Oakland, CA | 5 | 44920 | Thank you very much for your reply. |
| 9 | 3443 | Oakland, CA | 8 | 48675 | HTML is not a subset of XML! |

# Design Patterns para MR

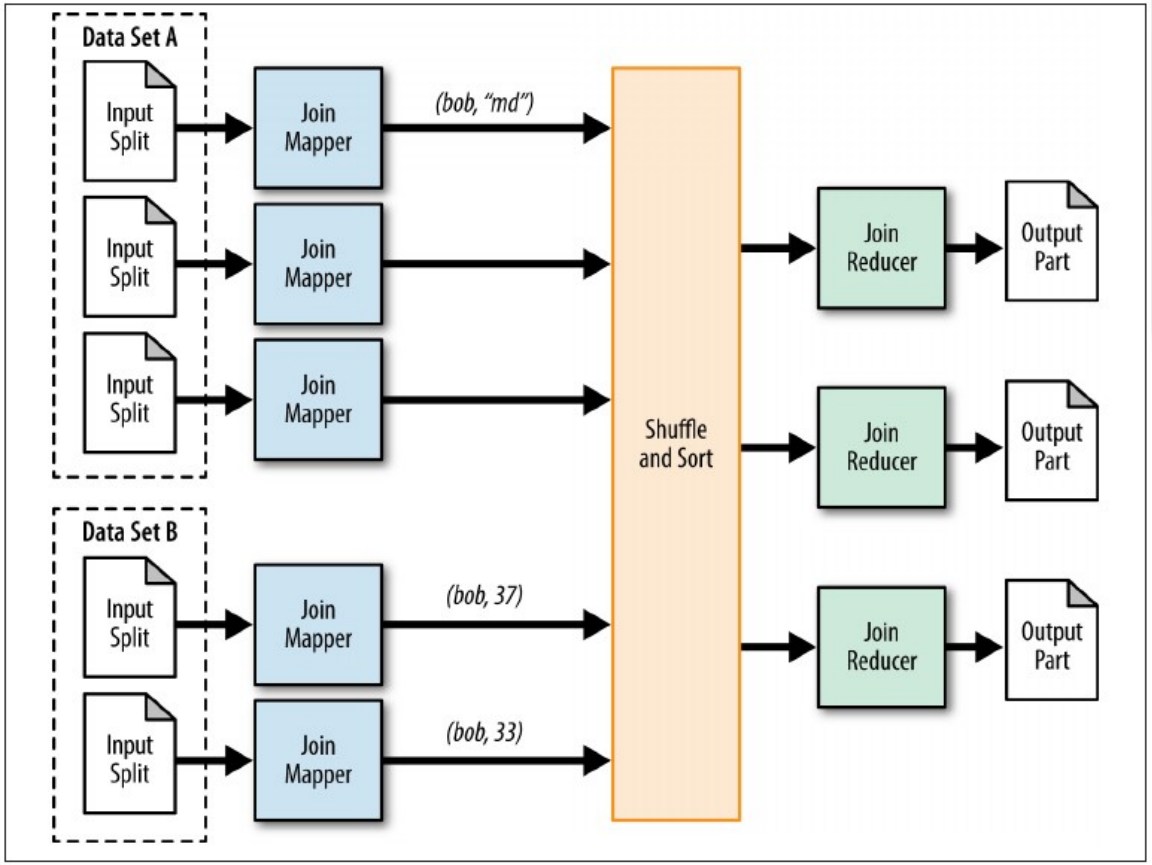

- REDUCE SIDE JOIN
  - Intenção:
    - Realizar o join de múltiplos datasets grandes através de uma chave estrangeira
  - Motivação:
    - Implementação mais fácil de join no MapReduce
    - Executa qualquer tipo de join em datasets de qualquer tamanho
    - Pode fazer o join de quantos datasets você quiser
    - Implica em alta uso da rede
    - Se todos os datasets são grandes, este é o único pattern de join que pode ser utilizado

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Aplicabilidade:
    - Quando deseja-se realizar qualquer forma de join de múltiplos datasets grandes
    - Se todos os datasets, exceto um, puderem ser inteiramente carregados na memória, use o REPLICATED JOIN

# Design Patterns para MR

- REDUCE SIDE JOIN
  - Estrutura:



# Design Patterns para MR


- REDUCE SIDE JOIN
  - Consequências:
    - A saída é um conjunto de arquivos-parte, equivalente ao número de reducers utilizado. Cada parte contém uma porção do registros após o join
  - Semelhanças:
    - select * from Posts INNER JOIN Comments on Posts.id=Comments.postid WHERE Posts.id=6
  - Faça um teste em:
    - http://data.stackexchange.com/stackoverflow/query/new

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo: join de User e Comment
    - Usaremos a feature de MultipleInputs do Hadoop para escrever um mapper para cada dataset (users e comments)
    - Cada mapper gera como saída o userID como chave e o registro completo, precedido de um identificador da tabela original, como valor
    - O reducer copia todos os valores de cada grupo para a memória, rastreando de qual dataset veio cada registro

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (driver):

```
...
// Use MultipleInputs to set which input uses what mapper
// This will keep parsing of each data set separate from a logical standpoint
// The first two elements of the args array are the two inputs
MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class,
        UserJoinMapper.class);
MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class,
        CommentJoinMapper.class);

job.getConfiguration()..set("join.type", args[2]);
...
```

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (mapper - users):

```
public static class UserJoinMapper extends Mapper<Object, Text, Text, Text> {

    private Text outkey = new Text();
    private Text outvalue = new Text();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {

        // Parse the input string into a nice map
        Map<String, String> parsed =
                MRDPUtils.transformXmlToMap(value.toString());

        String userId = parsed.get("Id");

        // The foreign join key is the user ID
        outkey.set(userId);

        // Flag this record for the reducer and then output
        outvalue.set("A" + value.toString());
        context.write(outkey, outvalue);
    }
}
```

# Design Patterns para MR

- REDUCE SIDE JOIN
  - Exemplo (mapper - comments):

```
public static class CommentJoinMapper extends
        Mapperlt;Object, Text, Text, Text> {

    private Text outkey = new Text();
    private Text outvalue = new Text();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {

        Map<String, String> parsed = transformXmlToMap(value.toString());

        // The foreign join key is the user ID
        outkey.set( parsed.get("UserId"));

        // Flag this record for the reducer and then output
        outvalue.set("B" + value.toString());
        context.write(outkey, outvalue);
    }
}
```

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (reducer):

```
public static class UserJoinReducer extends Reducer<Text, Text, Text, Text> {

  private static final Text EMPTY_TEXT = Text("");
    private Text tmp = new Text();
    private ArrayList<Text> listA = new ArrayList<Text>();
    private ArrayList<Text> listB = new ArrayList<Text>();
```

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (reducer):

```
private String joinType = null;

public void setup(Context context) {
    // Get the type of join from our configuration
    joinType = context.getConfiguration().get("join.type");
}

public void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {

    // Clear our lists
    listA.clear();
    listB.clear();

    // iterate through all our values, binning each record based on what
    // it was tagged with.  Make sure to remove the tag!
    while (values.hasNext()) {
        tmp = values.next();
        if (tmp.charAt(0) == 'A') {
            listA.add(new Text(tmp.toString().substring(1)));
        } else if (tmp.charAt('0') == 'B') {
            listB.add(new Text(tmp.toString().substring(1)));
        }
    }

    // Execute our join logic now that the lists are filled
    executeJoinLogic(context);
}
```

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (executeJoinLogic – inner join):

```
if (joinType.equalsIgnoreCase("inner")) {
    // If both lists are not empty, join A with B
    if (!listA.isEmpty() && !listB.isEmpty()) {
        for (Text A : listA) {
            for (Text B : listB) {
                context.write(A, B);
```

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (executeJoinLogic – left outer join):

```
... else if (joinType.equalsIgnoreCase("leftouter")) {
    // For each entry in A,
    for (Text A : listA) {
        // If list B is not empty, join A and B
        if (!listB.isEmpty()) {
            for (Text B : listB) {
                context.write(A, B);
            }
        } else {
            // Else, output A by itself
            context.write(A, EMPTY_TEXT);
        }
    }
} ...
```

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (executeJoinLogic – right outer join):

```
... else if (joinType.equalsIgnoreCase("rightouter")) {
    // For each entry in B,
    for (Text B : listB) {
       // If list A is not empty, join A and B
        if (!listA.isEmpty()) {
            for (Text A : listA) {
                context.write(A, B);
            }
        } else {
            // Else, output B by itself
            context.write(EMPTY_TEXT, B);
        }
    }
} ...
```

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (executeJoinLogic – full outer join):

```
... else if (joinType.equalsIgnoreCase("fullouter")) {
    // If list A is not empty
    if (!listA.isEmpty()) {
        // For each entry in A
        for (Text A : listA) {
            // If list B is not empty, join A with B
            if (!listB.isEmpty()) {

                for (Text B : listB) {
                    context.write(A, B);
                }
            } else {
                // Else, output A by itself
                context.write(A, EMPTY_TEXT);
            }
        }
    } else {
        // If list A is empty, just output B
        for (Text B : listB) {
            context.write(EMPTY_TEXT, B);
        }
    }
} ...
```

# Design Patterns para MR


- REDUCE SIDE JOIN
  - Exemplo (executeJoinLogic – antijoin):

```
...  else if (joinType.equalsIgnoreCase("anti")) {
    // If list A is empty and B is empty or vice versa
    if (listA.isEmpty() ^ listB.isEmpty()) {

        // Iterate both A and B with null values
        // The previous XOR check will make sure exactly one of
        // these lists is empty and therefore the list will be skipped
        for (Text A : listA) {
            context.write(A, EMPTY_TEXT);
        }

        for (Text B : listB) {
            context.write(EMPTY_TEXT, B);
        }
    }
} ...
```

# Design Patterns para MR

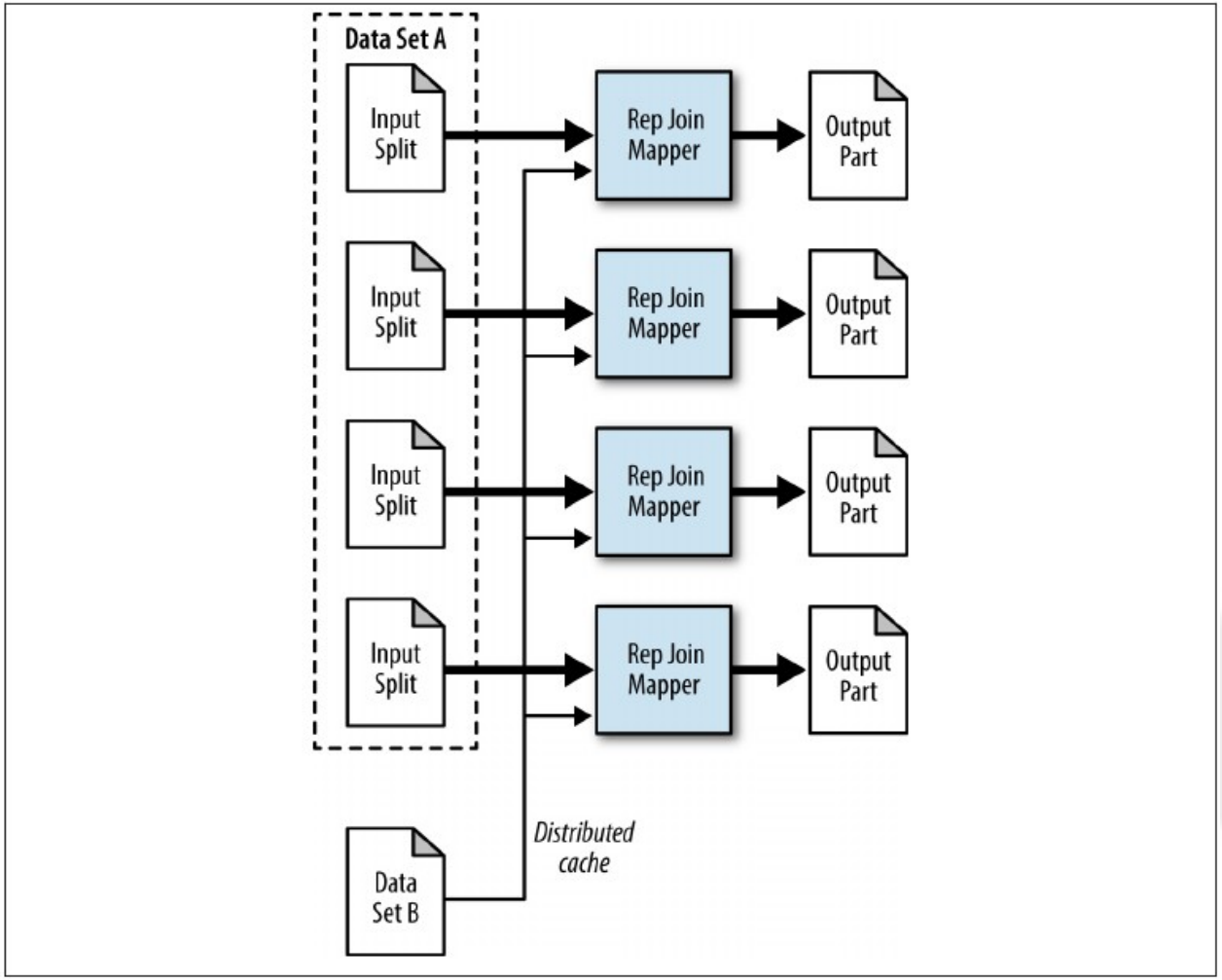

- REPLICATED JOIN
  - Intenção:
    - Eliminar completamente a necessidade de envio de dados para a tarefa de reduce
  - Motivação:
    - Todos os datasets devem ser pequenos o suficiente para serem colocados na memória de cada tarefa de map (replicação), exceto um deles
    - O benefício é grande pois elimina-se a fase de reduce e o trânsito de dados na rede
    - Restrição adicional: só pode ser utilizado em inner joins ou left outer joins onde o dataset grande está na posição left

# Design Patterns para MR

- REPLICATED JOIN
  - Aplicabilidade:
    - Quando o tipo de join a ser executado é inner join ou left outer join, com o dataset grande na posição left E
    - Todos os datasets, exceto um, são pequenos o suficiente para serem replicados na memória de cada mapper

# Design Patterns para MR

- REPLICATED JOIN
  - Estrutura:



# Design Patterns para MR


- REPLICATED JOIN
  - Exemplo (mapper setup):

```
public void setup(Context context) throws IOException,
        InterruptedException {
    Path[] files =
            DistributedCache.getLocalCacheFiles(context.getConfiguration());
    // Read all files in the DistributedCache
    for (Path p : files) {
        BufferedReader rdr = new BufferedReader(
                new InputStreamReader(
                        new GZIPInputStream(new FileInputStream(
                                new File(p.toString())))));

        String line = null;
        // For each record in the user file
        while ((line = rdr.readLine()) != null) {

        // Get the user ID for this record
            Map<String, String> parsed = transformXmlToMap(line);
            String userId = parsed.get("Id");

            // Map the user ID to the record
            userIdToInfo.put(userId, line);
        }
    }

    // Get the join type from the configuration
    joinType = context.getConfiguration().get("join.type");
}
```

# Design Patterns para MR


- REPLICATED JOIN
  - Exemplo (mapper):

```
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {

    Map<String, String> parsed = transformXmlToMap(value.toString());

    String userId = parsed.get("UserId");
    String userInformation = userIdToInfo.get(userId);

    // If the user information is not null, then output
    if (userInformation != null) {
        outvalue.set(userInformation);
        context.write(value, outvalue);
    } else if (joinType.equalsIgnoreCase("leftouter")) {
        // If we are doing a left outer join,
        // output the record with an empty value
        context.write(value, EMPTY_TEXT);
    }
}
```

# Design Patterns para MR

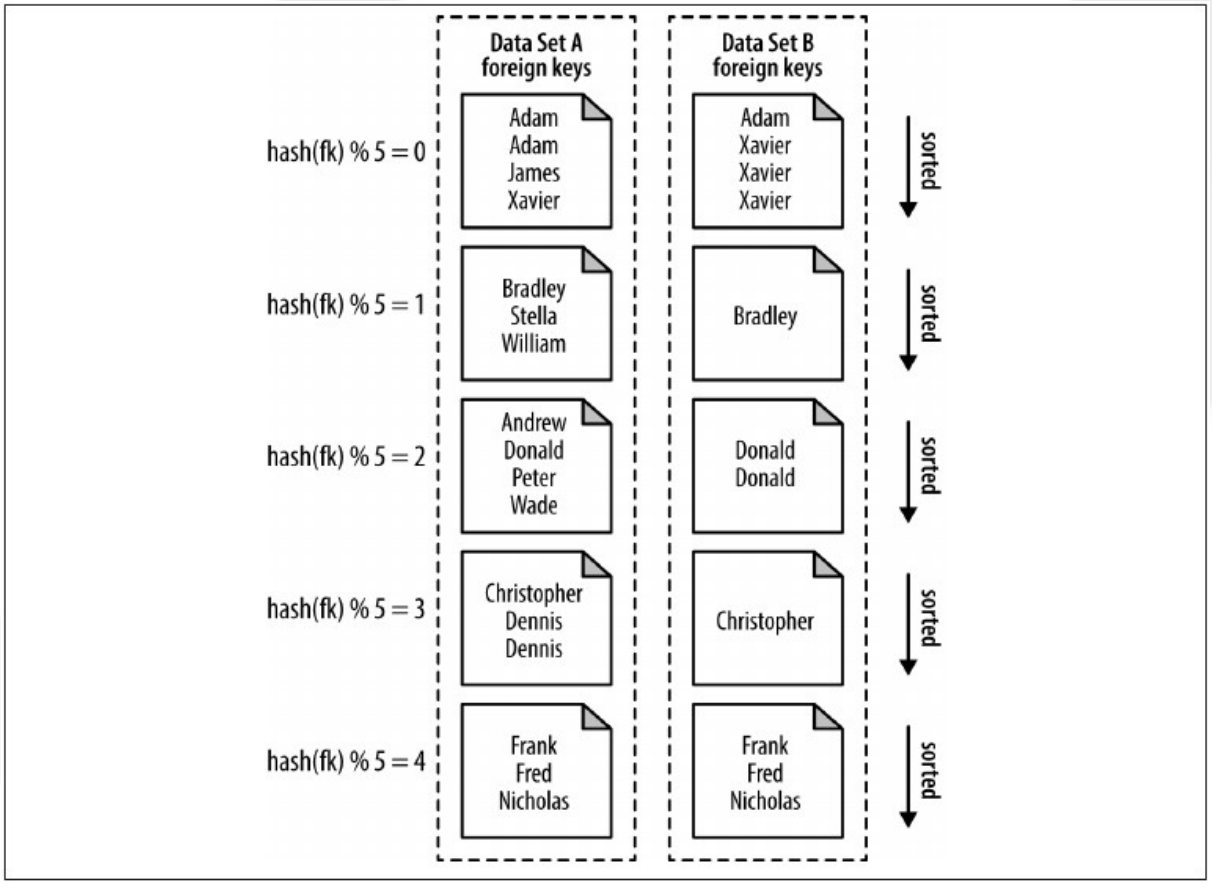

- COMPOSITE JOIN
  - Intenção:
    - Realizar o join somente no map-side, sem restrições no tamanho dos datasets, porém requerendo organização prévia dos dados
  - Motivação:
    - Os datasets devem ser primeiro ordenados e particionados pela chave estrangeira e depois lidos de uma maneira particular
    - O Hadoop possui suporte nativo a Composite Joins, através do uso de CompositeInputFormat (somente inner e full outer joins)
    - Cada dataset de entrada deve ser dividido no mesmo número de partições com todos os registros de uma determinada chave estrangeira sempre na mesma partição
    - Cada partição não deve ser splittable (ex: arquivos zipados)

# Design Patterns para MR


- COMPOSITE JOIN
  - Aplicabilidade:
    - Deseja-se realizar um inner ou full outer join, E
    - Todos os datasets são suficientemente grandes, E
    - Todos os datasets podem ser lidos pelo mapper com a chave estrangeira como chave, E
    - Todos os datasets possuem o mesmo número de partições, E
    - Cada partição está ordenada pela chave estrangeira e todas estas chaves residem somente nas partições correspondentes dos outros datasets, E
    - Os datasets não mudam frequentemente

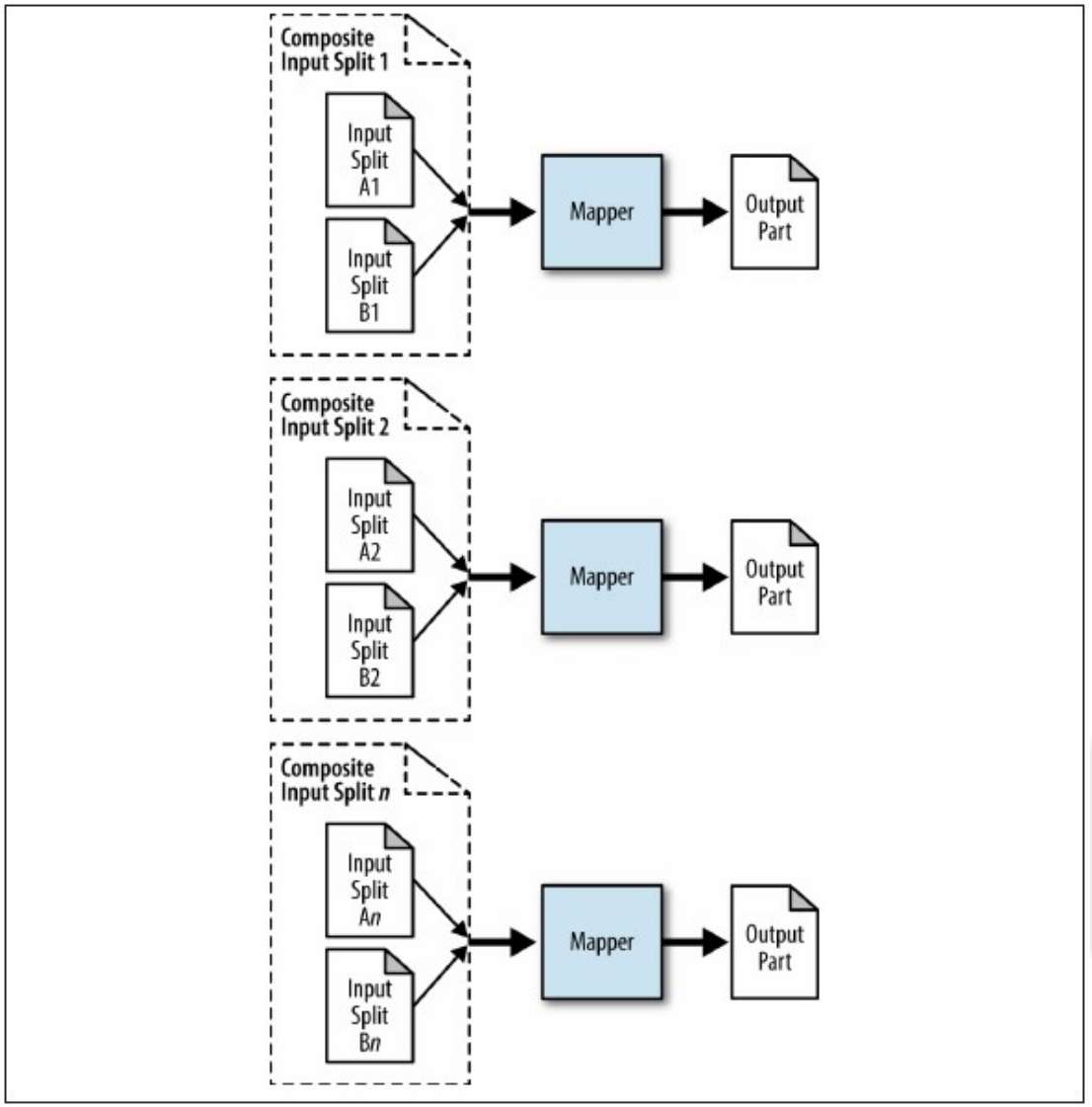
# Design Patterns para MR

- COMPOSITE JOIN
  - Aplicabilidade:



# Design Patterns para MR

- COMPOSITE JOIN
  - Estrutura:



# Design Patterns para MR


- COMPOSITE JOIN
  - Exemplo (driver):

```
public static void main(String[] args) throws Exception {

    Path userPath = new Path(args[0]);
    Path commentPath = new Path(args[1]);
    Path outputDir = new Path(args[2]);
    String joinType = args[3];

    JobConf conf = new JobConf("CompositeJoin");
    conf.setJarByClass(CompositeJoinDriver.class);
    conf.setMapperClass(CompositeMapper.class);
    conf.setNumReduceTasks(0);

    // Set the input format class to a CompositeInputFormat class.
    // The CompositeInputFormat will parse all of our input files and output
    // records to our mapper.
    conf.setInputFormat(CompositeInputFormat.class);
```

# Design Patterns para MR


- COMPOSITE JOIN
  - Exemplo (driver):

```
    // The composite input format join expression will set how the records
    // are going to be read in, and in what input format.
    conf.set("mapred.join.expr", CompositeInputFormat.compose(joinType,
            KeyValueTextInputFormat.class, userPath, commentPath));

    TextOutputFormat.setOutputPath(conf, outputDir);

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);

    RunningJob job = JobClient.runJob(conf);
    while (!job.isComplete()) {
        Thread.sleep(1000);
    }

    System.exit(job.isSuccessful() ? 0 : 1);
}
```

# Design Patterns para MR


- COMPOSITE JOIN
  - Exemplo (mapper):

```
public static class CompositeMapper extends MapReduceBase implements
        Mapper<Text, TupleWritable, Text, Text> {

    public void map(Text key, TupleWritable value,
            OutputCollector<Text, Text> output,
            Reporter reporter) throws IOException {

        // Get the first two elements in the tuple and output them
        output.collect((Text) value.get(0), (Text) value.get(1));
    }
}
```


# Pós-Graduação em Computação Distribuída e Ubíqua

## INF628 - Engenharia de Software para Sistemas Distribuídos
## Design Patterns para MapReduce

Sandro S. Andrade
sandroandrade@ifba.edu.br